



A Siemens Business

Tessent® Shell Reference Manual

Software Version 2018.3

August 2018

Document Revision 10

© 2011-2018 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

Note - Viewing PDF files within a web browser causes some links not to function (see [MG595892](#)).
Use HTML for full navigation.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: mentor.com/trademarks.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

End-User License Agreement: You can print a copy of the End-User License Agreement from: mentor.com/eula.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: mentor.com
Support Center: support.mentor.com

Send Feedback on Documentation: support.mentor.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
10	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Tessent Release Notes</i> for this product are reflected in this document. Approved by Ron Press.	Released Aug 2018
9	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Tessent Release Notes</i> for this product are reflected in this document. Approved by Ron Press.	Released May 2018
8	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Tessent Release Notes</i> for this product are reflected in this document. Approved by Ron Press.	Released Mar 2018
7	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Tessent Release Notes</i> for this product are reflected in this document. Approved by Ron Press.	Released Dec 2017

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Mentor Graphics Technical Publication's source. For specific topic authors, contact Mentor Graphics Technical Publication department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available at the following URL:

<http://support.mentor.com>

Table of Contents

Revision History

Chapter 1

Introduction.....	53
Overview	53
Product Introduction	56
Tessent BoundaryScan	56
Tessent Diagnosis	58
Tessent FastScan and Tessent TestKompress.....	58
Tessent IJTAG	61
Tessent LogicBIST	61
Tessent MemoryBIST.....	62
Tessent MissionMode	63
Tessent Scan	63
Tessent ScanPro	64
Tessent SiliconInsight.....	65
Tessent DesignEditor	65

Chapter 2

Tool Invocations	67
Tool Invocation Command Descriptions.....	68
create_skeleton_design.....	69
stil2mgc.....	71
tessent	75

Chapter 3

Command Dictionary (A - D)	81
Command Descriptions	82
add_ambiguous_paths.....	94
add_atpg_constraints	96
add_atpg_functions.....	100
add_bist_capture_range	103
add_black_boxes.....	105
add_browser_data	109
add_capture_handling	111
add_cdp_test	114
add_cell_constraints	115
add_cell_models	122
add_chain_masks	124
add_clocks	128
add_config_element	138
add_config_message.....	143

add_config_tab	145
add_control_points	146
add_core_instances	148
add_dft_clock_enables	159
add_dft_clock_mux	164
add_dft_control_points	169
add_dft_modal_connections	176
add_dft_signals	187
add_display_callout	223
add_display_data	226
add_display_instances	228
add_display_path	232
add_drcViolation	234
add_edt_blocks	241
add_edt_configurations	244
add_false_paths	246
add_fault_sites	253
add_faults	255
add_icl_ports	268
add_icl_scan_interfaces	277
add_iddq_exceptions	279
add_ijtag_logical_connection	281
add_input_constraints	286
add_layout_core_information	293
add_lfsr_connections	295
add_lfsr_taps	297
add_lfsrs	299
add_lists	302
add_loadboard_loopback_pairs	304
add_misrs	306
add_nofaults	308
add_nonscan_instances	312
add_notest_points	315
add_observe_points	317
add_output_masks	319
add_primary_inputs	321
add_primary_outputs	324
add_processors	326
add_read_controls	329
add_register_value	331
add_rtl_to_gates_mapping	333
add_scan_chains	336
add_scan_groups	339
add_scan_instances	341
add_scan_mode	343
add_scan_segments	348
add_simdut_fault	350
add_simulation_context	352
add_simulation_forces	354

Table of Contents

add_synchronous_clock_group	356
add_tied_signals	358
add_to_collection	360
add_write_controls	362
analyze_atpg_constraints	364
analyze_bus	367
analyze_compression	371
analyze_control_signals	393
analyze_drcViolation	396
analyze_fault	398
analyze_graybox	404
analyze_restrictions	408
analyze_scan_chains	411
analyze_simulation_mismatches	412
analyze_test_points	420
analyze_wrapper_cells	425
analyze_xbounding	428
annotate_diagnosis	430
append_to_collection	432
apply_specification_defaults	434
catch_output	437
check_design_rules	439
check_synthesis	441
check_testbench_simulations	444
close_layout	450
close_pattern_set	451
close_tsdb	453
close_visualizer	455
compare_collections	456
compress_layout	458
compress_patterns	459
copy_collection	462
copy_module	464
copy_simulation_context	466
create_bisr_segment_order_file	468
create_capture_procedures	469
create_connections	473
create_dft_specification	477
create_diagnosis_patterns	491
create_feature_statistics	493
create_flat_model	496
create_icl_verification_patterns	498
create_initialization_patterns	502
create_instance	504
create_layout	508
create_module	516
create_net	519
create_patterns	521
create_patterns_specification	530

create_pin	533
create_port	535
create_scan_chain_family	537
create_silicon_insight_setup_specification	541
delete_atpg_constraints	542
delete_atpg_functions	544
delete_bist_capture_ranges	546
delete_black_boxes	547
delete_browser_data	549
delete_capture_handling	551
delete_capture_procedures	553
delete_cdp_test	555
delete_cell_constraints	556
delete_cell_library	558
delete_cell_models	559
delete_chain_masks	561
delete_clocks	562
delete_config_element	563
delete_config_messages	565
delete_config_tabs	567
delete_connections	568
delete_core_descriptions	570
delete_core_instances	572
delete_design	575
delete_dfm	576
delete_dft_clock_enables	578
delete_dft_clock_muxes	580
delete_dft_control_points	582
delete_dft_modal_connections	584
delete_dft_signals	587
delete_display_callout	589
delete_display_data	590
delete_display_instances	592
delete_display_path	593
delete_edt_blocks	594
delete_edt_configurations	597
delete_false_paths	599
delete_fault_sites	602
delete_faults	606
delete_flat_model	618
delete_icl_modules	619
delete_icl_ports	620
delete_icl_scan_interfaces	621
delete_iddq_exceptions	622
delete_ijtag_logical_connection	623
delete_input_constraints	625
delete_instances	628
delete_iprocs	629
delete_layout_core_information	631

Table of Contents

delete_layout_verification	632
delete_lfsr_connections	634
delete_lfsr_taps	635
delete_lfsrs	636
delete_lists	637
delete_loadboard_loopback_pairs	638
delete_misrs	640
delete_multicycle_paths	641
delete_nets	644
delete_nofaults	646
delete_nonscan_instances	649
delete_notest_points	651
delete_output_masks	654
delete_patterns	656
delete_pins	659
delete_ports	661
delete_primary_inputs	663
delete_primary_outputs	665
delete_processors	667
delete_read_controls	669
delete_register_value	670
delete_rtl_to_gates_mapping	671
delete_scan_chain_families	673
delete_scan_chains	674
delete_scan_elements	675
delete_scan_groups	676
delete_scan_instances	677
delete_scan_modes	679
delete_simdut_fault	680
delete_simulation_contexts	681
delete_simulation_forces	683
delete_synchronous_clock_group	685
delete_test_points	687
delete_tied_signals	689
delete_write_controls	691
diagnose_failures	692
display_diagnosis_report	696
display_message	698
display_specification	700
dofile	702

Chapter 4	
Command Dictionary (E - Q)	705
Command Descriptions	706
echo	718
execute_cdp_test	720
execute_gpib_command	724
execute_tester_command	726

exit	728
expand_compressed_patterns	729
extract_icl	730
extract_sdc	733
filter_collection	735
find_design_names	737
foreach_in_collection	743
format_dictionary	746
get_attribute_list	750
get_attribute_option	753
get_attribute_value_list	755
get_attributed_objects	759
get_auxiliary_pins	761
get_boundary_scan_port_option	764
get_cdp_test_list	765
get_clock_option	766
get_clocks	771
get_common_parent_instance	774
get_config_elements	775
get_config_messages	779
get_config_selected_nodes	782
get_config_value	784
get_context	798
get_current_design	802
get_current_mode	804
get_current_silicon_insight_setup	805
get_current_simulation_context	806
get_defaults_value	807
get_design_level	809
get_design_objects	810
get_design_sources	813
get_dfm_rules	815
get_dft_cell	817
get_dft_info_dictionary	827
get_dft_signal	834
get_dft_specification_requirement	840
get_drc_handling	842
get_equivalent_editable_node	843
get_fanins	845
get_fanouts	848
get_fault_type	851
get_gate_pins	852
get_icl_extraction_options	857
get_icl_fanins	858
get_icl_fanouts	860
get_icl_instances	862
get_icl_modules	866
get_icl_module_parameter_list	869
get_icl_module_parameter_value	870

Table of Contents

get_icl_objects	871
get_icl_pins	877
get_icl_ports	881
get_icl_scan_interface_list	884
get_icl_scan_interface_port_list	886
get_icl_scope	888
get_iclock_list	890
get_iclock_option	892
get_ijtag_instances	896
get_ijtag_instance_option	898
get_ijtag_retargeting_options	900
get_input_constraints	903
get_insertion_option	907
get_insert_test_logic_option	909
get_instances	912
get_instrument_dictionary	918
get_instrument_parent_icl_module_list	922
get_iproc_argument_default	923
get_iproc_argument_list	925
get_iproc_body	926
get_iproc_list	927
get_layout_core_instance	928
get_license_queue_timeout	929
get_loadboard_loopback_option	930
get_logfile	931
get_logical_library_list	932
get_memory_instances	933
get_memory_instance_option	935
get_module_parameter_list	938
get_module_parameter_value	940
get_modules	941
get_multiprocessing_option	946
get_name_list	948
get_nets	950
get_open_pattern_set	953
get_pattern_cycle_count	954
get_pattern_set_data	955
get_pattern_set_list	963
get_pattern_set_option	964
get_pins	966
get_ports	971
get_procfile_name	975
get_read_verilog_option	976
get_resource	977
get_run_synthesis_options	979
get_scan_chain_families	981
get_scan_chain_option	982
get_scan_elements	983
get_scan_modes	987

get_scratch_directory	988
get_silicon_insight_job_status	989
get_silicon_insight_option	990
get_simulation_context_list	991
get_simulation_library_sources	993
get_simulation_option	995
get_simulation_value_list	997
get_single_attribute_value	999
get_single_name	1001
get_static_dft_signal_icall	1003
get_synchronous_clock_groups	1005
get_system_mode	1006
get_tcl_shell_option	1007
get_test_end_icall_list	1008
get_test_point_type	1009
get_test_points	1010
get_test_setup_icall_list	1014
get_testbench_simulation_options	1015
get_timeplate_list	1017
get_tool_info	1018
get_tool_option	1020
get_trace_flat_model_option	1021
get_transcript_style	1024
get_tsdb_info	1026
get_tsdb_list	1032
get_tsdb_output_directory	1033
get_validated_objects	1035
get_write_patterns_options	1038
get_xclock_handling	1040
help	1041
history	1043
iApply	1045
iCall	1048
iClock	1052
iClockOverride	1054
iComparePort	1057
identify_redundant_faults	1060
iForcePort	1064
iMerge	1068
import_clocks	1070
import_dfm	1071
import_patterns_from_svf	1073
import_scan_mode	1076
index_collection	1079
iNote	1081
insert_test_logic	1086
intercept_connection	1089
iOverrideScanInterface	1097
iPDLLevel	1099

Table of Contents

iPrefix	1100
iProc	1102
iProcsForModule	1104
iPulseClock	1106
iRead	1108
iRelease	1110
iReset	1111
iRunLoop	1113
is_collection	1116
iTake	1118
iTopProc	1120
iUseProcNameSpace	1122
iWrite	1124
launch_sid_tester	1126
lock_current_registration	1128
macrotest	1129
mark_display_instances	1136
merge_cdp	1138
migrate_layout	1139
move_config_element	1140
move_connections	1145
no_transcript	1152
open_layout	1155
open_pattern_set	1159
open_tsdb	1167
open_visualizer	1169
order_patterns	1171
printenv	1175
process_dft_specification	1176
process_patterns_specification	1181
Chapter 5	
Command Dictionary (R)	1187
Command Descriptions	1188
range_collection	1200
read_cell_library	1202
read_config_data	1203
read_core_descriptions	1206
read_cpf	1208
read_def	1211
read_design	1214
read_failures	1220
read_fault_sites	1228
read_faults	1238
read_flat_model	1249
read_icl	1253
read_lef	1255
read_liberty	1256

read_modelfile	1257
read_patterns	1261
read_procfile	1271
read_sdc	1276
read_sdf	1283
read_upf	1286
read_verilog	1288
read_vhdl	1294
read_visualizer_preferences	1299
read_window_contents	1300
register_attribute	1302
register_callback	1306
register_drc	1315
register_drc_class	1351
register_static_dft_signal_names	1352
register_tcl_command	1356
remove_from_collection	1370
rename_instance	1372
rename_net	1374
replace_instances	1376
rename_module	1378
report_aborted_faults	1379
report_atpg_constraints	1382
report_atpg_functions	1383
report_atpg_timing	1384
report_attributes	1387
report_bISR_repair_register_icl_instances	1395
report_bist_capture_ranges	1396
report_black_boxes	1397
report_boundary_scan_port_options	1399
report_bus_data	1401
report_bypass_chains	1404
report_capture_handling	1407
report_capture_procedures	1409
report_cell_constraints	1415
report_cell_models	1418
report_chain_masks	1420
report_clock_controller_pins	1421
report_clock_controls	1422
report_clock_domains	1424
report_clock_gating	1429
report_clocks	1431
report_compactor_connections	1436
report_config_data	1439
report_config_messages	1444
report_config_syntax	1447
report_context	1451
report_control_signals	1453
report_core_descriptions	1455

Table of Contents

report_core_instance_parameters	1458
report_core_instances	1460
report_core_parameters	1463
report_design_sources	1465
report_dfm_rules.	1467
report_dft_clock_enables	1469
report_dft_clock_muxes	1471
report_dft_control_points	1472
report_dft_modal_connections	1474
report_dft_signal_names.	1478
report_dft_signals	1479
report_diagnosis	1481
report_display_instances.	1482
report_drc_rules	1485
report_edt_abort_analysis.	1492
report_edt_blocks	1509
report_edt_configurations.	1510
report_edt_finder	1516
report_edt_instances	1521
report_edt_lockup_cells	1524
report_edt_pins	1529
report_environment.	1532
report_external_simulator.	1534
report_failures	1535
report_false_paths.	1539
report_fault_sites	1547
report_faults	1555
report_feedback_paths	1569
report_flattener_rules	1572
report_gates.	1574
report_graybox_statistics	1599
report_icl_extraction_options.	1601
report_icl_modules	1602
report_iclock	1606
report_id_stamp	1608
report_iddq_exceptions.	1610
report_ijtag_instances	1611
report_ijtag_logical_connections	1613
report_ijtag_retargeting_options.	1615
report_input_constraints	1618
report_insert_test_logic_options.	1621
report_iprocs	1622
report_layout_core_information.	1623
report_layout_files	1625
report_layout_rules	1627
report_lbist_configuration	1632
report_lbist_pins	1634
report_lfsr_connections	1636
report_lfsrs	1637

report_licenses	1639
report_lists	1640
report_loadboard_loopback_pairs	1641
report_loops	1642
report_lpct_configuration	1644
report_lpct_pins	1645
report_measure_cycles	1647
report_memory_cluster_configuration	1652
report_memory_identification	1658
report_memory_instances	1661
report_memory_repair_groups	1663
report_mismatch_sources	1665
report_misrs	1669
report_misr_connections	1671
report_module_matching	1673
report_module_matching_options	1676
report_multicycle_paths	1678
report_multiprocessing_options	1684
report_nofaults	1686
report_noscan_cells	1688
report_noscan_models	1690
report_notest_points	1691
report_output_masks	1693
report_pattern_filtering	1694
report_pattern_sets	1696
report_patterns	1698
report_power_data	1709
report_power_metrics	1711
report_primary_inputs	1717
report_primary_outputs	1719
report_procedures	1721
report_processors	1725
report_read_controls	1728
report_read_verilog_options	1729
report_register_value	1730
report_resources	1733
report_rtl_to_gates_mapping	1734
report_run_synthesis_options	1736
report_scan_cells	1737
report_scan_chains	1745
report_scan_elements	1750
report_scan_enable	1753
report_scan_groups	1755
report_scan_models	1756
report_scan_modes	1757
report_scan_polygons	1758
report_scan_segments	1762
report_scan_volume	1764
report_seq_transparent_procedures	1768

Table of Contents

report_sequential_fault_depth	1770
report_shift_registers	1772
report_silicon_insight_result	1775
report_simdut_faults	1777
report_simulation_contexts	1778
report_simulation_forces	1780
report_simulation_library_sources	1782
report_static_dft_signal_settings	1783
report_statistics	1785
report_synchronous_clock_groups	1804
report_tcd_ldb_validation	1806
report_tcl_shell_options	1808
report_test_end_icall	1809
report_test_logic	1810
report_test_points	1811
report_test_point_statistics	1815
report_test_setup_icall	1817
report_test_stimulus	1818
report_testbench_simulation_options	1827
report_tied_signals	1828
report_timeplate	1830
report_udfm_statistics	1832
report_version_data	1836
report_wrapper_cells	1837
report_write_controls	1841
report_write_patterns_options	1842
report_xbounding	1844
reset_attribute_value	1846
reset_au_faults	1848
reset_bypass_chains	1850
reset_compactor_connections	1851
reset_design	1852
reset_di_faults	1853
reset_open_pattern_set	1856
reset_state	1857
reset_static_dft_signal_values	1858
restore_design	1861
run_synthesis	1862
run_testbench_simulations	1871
Chapter 6	
Command Dictionary (S - Z)	1887
Command Descriptions	1888
select_display_instances	1902
set_abort_limit	1904
set_atpg_fill	1906
set_atpg_limits	1907
set_atpg_timing	1910

set_attribute_options	1913
set_attribute_value	1918
set_au_analysis	1923
set_bidi_gating	1924
set_bist_chain_test	1927
set_bist_debug	1928
set_bist_trace	1932
set_boundary_scan_port_options	1935
set_bus_handling	1937
set_bus_simulation	1940
set_bypass_chains	1942
set_capture_clock	1945
set_capture_handling	1948
set_capture_procedures	1950
set_cell_library_options	1953
set_cell_model_mapping	1957
set_chain_test	1959
set_checkpointing_options	1963
set_clock_controller_pins	1965
set_clock_controls	1969
set_clock_gating	1972
set_clock_gating_enable	1975
set_clock_off_simulation	1977
set_clock_options	1979
set_clock_restriction	1982
set_compactor_connections	1989
set_config_value	1992
set_contention_check	1996
set_context	2001
set_core_instance_parameters	2007
set_current_design	2009
set_current_edt_block	2014
set_current_edt_configuration	2017
set_current_mode	2019
set_current_silicon_insight_setup	2021
set_current_simulation_context	2023
set_decision_order	2025
set_dedicated_wrapper_cell_options	2026
set_defaults_value	2029
set_design_include_directories	2033
set_design_level	2035
set_design_macros	2038
set_design_sources	2039
set_dft_enable_options	2043
set_dft_specification_requirements	2045
set_diagnosis_options	2049
set_display	2061
set_dofile_abort	2063
set_drc_handling	2065

Table of Contents

set_driver_restriction	2073
set_edt_abort_analysis_options	2074
set_edt_finder	2075
set_edt_instances	2077
set_edt_mapping	2081
set_edt_options	2084
set_edt_pins	2095
set_edt_power_controller	2106
set_external_capture_options	2110
set_external_simulator	2113
set_fails_report	2115
set_failure_mapping_options	2116
set_fault_mode	2117
set_fault_sampling	2119
set_fault_subclass_analysis	2121
set_fault_type	2124
set_flattener_rule_handling	2133
set_gate_level	2136
set_gate_report	2138
set_gzip_options	2172
set_icl_extraction_options	2174
set_icl_scan_interface_ports	2175
set_iddq_checks	2178
set_ijtag_instance_options	2184
set_ijtag_retargeting_options	2186
set_insertion_options	2193
set_insert_test_logic_options	2199
set_internal_fault	2202
set_internal_name	2203
set_io_mask	2204
set_latch_handling	2205
set_layout_core_instance	2206
set_lbist_controller_options	2207
set_lbist_instances	2211
set_lbist_pins	2213
set_lbist_power_controller_options	2220
set_learn_report	2222
set_lfsr_seed	2224
set_lfsrs	2225
set_license_queue_timeout	2227
set_list_file	2229
set_logfile_handling	2230
set_logical_design_libraries	2232
set_loop_handling	2234
set_lpct_condition_bits	2237
set_lpct_controller	2239
set_lpct_instances	2242
set_lpct_pins	2243
set_macrotest_options	2251

set_memory_identification_options	2253
set_memory_instance_options	2255
set_misr_connections	2259
set_module_matching_options	2261
set_multiple_detection	2263
set_multiprocessing_options	2267
set_net_dominance	2275
set_net_resolution	2277
set_number_shifts	2278
set_observation_point	2280
set_output_masks	2282
set_parallel_load_subchains	2285
set_pathdelay_holdpi	2287
set_pattern_buffer	2288
set_pattern_classification	2290
set_pattern_filtering	2291
set_pattern_source	2301
set_pattern_type	2310
set_physical_translation	2316
set_possible_credit	2318
set_power_control	2319
set_power_metrics	2324
set_procedure_cycle_checking	2329
set_procedure_retargeting_options	2330
set_procfile_name	2333
set_quick_synthesis_options	2335
set_ram_initialization	2338
set_random_atpg	2339
set_random_clocks	2340
set_random_patterns	2342
set_read_verilog_options	2343
set_register_value	2344
set_relevant_coverage	2346
set_run_synthesis_options	2350
set_scan_chain_options	2352
set_scan_enable	2354
set_scan_insertion_options	2358
set_scan_signals	2362
set_screen_display	2365
set_shadow_check	2366
set_shift_register_identification	2368
set_silicon_insight_option	2370
set_simulation_library_sources	2372
set_simulation_options	2375
set_skewed_load	2379
set_split_capture_cycle	2380
set_stability_check	2382
set_static_dft_signal_values	2386
set_static_learning	2390

Table of Contents

set_system_mode	2392
set_tcl_shell_options	2394
set_test_end_icall	2396
set_test_logic	2398
set_test_point_analysis_options	2400
set_test_point_insertion_options	2410
set_test_point_types	2413
set_test_setup_icall	2415
set_tb_simulation_options	2420
set_tied_signals	2423
set_timing_exceptions_handling	2425
set_tla_loop_handling	2431
set_tool_options	2433
set_trace_flat_model_options	2435
set_trace_report	2438
set_transcript_style	2439
set_transient_detection	2442
set_transition_holdpi	2444
set_tristate_gating	2445
set_tsdb_output_directory	2447
set_visualizer_logging	2449
set_visualizer_preferences	2450
set_wrapper_analysis_options	2453
set_write_patterns_options	2457
set_xbounding_options	2460
set_xclock_handling	2464
set_z_handling	2466
set_zhold_behavior	2468
setenv	2469
shutdown_sid_tester	2470
simulate_clock_pulses	2471
simulate_forces	2473
simulate_patterns	2475
sizeof_collection	2478
sort_collection	2480
start_silicon_insight	2482
stop_silicon_insight	2483
system	2484
trace_flat_model	2486
uncompress_layout	2494
uniquify_instances	2495
unmark_display_instances	2497
unregister_attribute	2499
unregister_static_dft_signal_names	2501
unregister_tcl_command	2502
unselect_display_instances	2503
unsetenv	2505
update_implication_detections	2506
verify_patterns	2508

write_atpg_setup	2512
write_cell_library	2515
write_config_data	2519
write_core_description	2523
write_core_timing_constraints	2525
write_design	2526
write_design_import_script	2536
write_design_source_dictionary	2538
write_diagnosis	2540
write_edt_files	2543
write_failing_paths	2557
write_failures	2560
write_fault_sites	2565
write_faults	2569
write_flat_model	2581
write_icl	2584
write_loops	2586
write_memory_repair_dictionary	2587
write_modelfile	2588
write_patterns	2589
write_primary_inputs	2610
write_primary_outputs	2611
write_procedure_testbench	2612
write_procfile	2616
write_scan_order	2618
write_scan_setup	2626
write_test_point_dofile	2628
write_tsdb_data	2630
write_visualizer_dofile	2634
write_visualizer_preferences	2636
write_window_contents	2637
Chapter 7	
Design Rule Checking	2639
Design Rule Overview	2640
Scan Insertion Checking	2640
How to Troubleshoot Rules Violations	2641
Rules Settings	2641
How to Turn on ATPG Analysis	2641
How to Debug DRC Violations in DFTVisualizer	2641
How to Set the Level of Gate Data	2646
How to Set the Gate Information Type	2648
How to Report Gate Data	2648
Flattening Rule Violations	2652
Design Rule Output	2652
Design Rules	2653
RAM Rules (A Rules)	2666
A1	2666

Table of Contents

A2	2667
A3	2668
A4	2668
A5	2669
A6	2670
A7	2670
A9	2671
A10	2672
A11	2672
A12	2673
A13	2674
A14	2675
A15	2675
A16	2676
A17	2676
A18	2677
Assertion Rules (ASSERT Rules)	2680
Enforcement of Assertions	2680
ASSERT1	2680
ASSERT2	2682
ASSERT3	2683
BIST Rules (B Rules)	2686
B2	2686
B4	2686
Clock Rules (C Rules)	2687
Clock Terminology	2687
The ATPG Analysis Option	2692
C1	2692
C2	2697
C3	2698
C4	2705
C5	2711
C6	2713
C7	2716
C8	2717
C9	2718
C10	2720
C16	2721
C17	2722
C19	2723
C21	2724
C22	2725
C23	2728
C24	2729
C25	2730
Scan Cell Data Rules (D Rules)	2732
D1	2732
D2	2737
D3	2738

Table of Contents

D4	2738
D5	2739
D6	2740
D7	2741
D8	2742
D9	2743
D10	2744
D12	2746
Pre-DFT Clock Rules (DFT_C Rules)	2747
Simulation Contexts for Pre-DFT DRCs	2747
DFT_C1	2751
DFT_C2	2754
DFT_C3	2755
DFT_C4	2756
DFT_C5	2757
DFT_C6	2758
DFT_C7	2760
DFT_C8	2761
DFT_C9	2762
DFT_C10	2765
DFT_C11	2771
DFT_C12	2772
DFT_C13	2773
Extra Rules (E Rules)	2775
E1	2775
E2	2776
E3	2776
E4	2777
E5	2780
E6	2782
E7	2783
E8	2783
E9	2784
E10	2785
E11	2790
E12	2791
E13	2791
E14	2792
E15	2795
EDT Finder Rules (F Rules)	2796
F3	2796
F4	2797
F5	2797
F6	2798
F7	2800
F8	2800
F9	2806
F10	2807
F11	2809

Table of Contents

F12	2809
F13	2809
F14	2810
F15	2811
F16	2811
F17	2812
F18	2813
F19	2813
F20	2814
F21	2816
F22	2817
Flattening Rules (FN, FP, and FG Rules)	2821
FN1	2822
FN2	2822
FN3	2822
FN4	2822
FN5	2823
FN6	2823
FN7	2823
FN8	2823
FN9	2824
FP1	2824
FP2	2824
FP3	2824
FP4	2824
FP5	2825
FP6	2825
FP7	2825
FP8	2825
FP9	2826
FP10	2826
FP11	2826
FP12	2826
FP13	2826
FG1	2827
FG2	2827
FG3	2827
FG4	2827
FG5	2828
FG6	2828
FG7	2828
FG8	2828
General Rules (G Rules)	2829
G1	2829
G2	2829
G3	2830
G4	2830
G5	2830
G6	2831

Table of Contents

G7	2831
G8	2831
G9	2832
G10	2832
G11	2833
G12	2833
ICL Extraction Rules (I Rules)	2834
I1	2834
I2	2835
I3	2837
I4	2838
I5	2839
I6	2840
I7	2842
ICL Semantic Rules (ICL Rules)	2843
ICL1	2847
ICL2	2848
ICL3	2848
ICL4	2848
ICL5	2849
ICL6	2849
ICL7	2849
ICL8	2849
ICL9	2850
ICL10	2850
ICL11	2850
ICL12	2851
ICL13	2851
ICL14	2851
ICL15	2851
ICL16	2852
ICL17	2852
ICL18	2852
ICL19	2853
ICL20	2853
ICL21	2853
ICL22	2853
ICL23	2854
ICL24	2854
ICL25	2854
ICL26	2854
ICL27	2855
ICL28	2855
ICL29	2855
ICL30	2855
ICL31	2856
ICL32	2856
ICL33	2856
ICL34	2857

Table of Contents

ICL35	2857
ICL36	2857
ICL37	2857
ICL38	2858
ICL39	2858
ICL40	2858
ICL41	2859
ICL42	2859
ICL43	2859
ICL44	2860
ICL45	2860
ICL48	2860
ICL49	2860
ICL50	2861
ICL51	2861
ICL52	2862
ICL53	2862
ICL54	2862
ICL55	2863
ICL56	2863
ICL57	2863
ICL58	2863
ICL59	2864
ICL60	2864
ICL61	2864
ICL62	2865
ICL63	2865
ICL64	2865
ICL65	2865
ICL66	2866
ICL67	2866
ICL68	2866
ICL70	2866
ICL71	2867
ICL75	2870
ICL76	2870
ICL77	2870
ICL78	2871
ICL79	2871
ICL80	2872
ICL81	2872
ICL82	2872
ICL83	2873
ICL84	2873
ICL85	2873
ICL86	2873
ICL87	2874
ICL88	2874
ICL89	2874

Table of Contents

ICL90	2875
ICL91	2876
ICL92	2876
ICL93	2876
ICL94	2877
ICL95	2877
ICL96	2877
ICL97	2877
ICL98	2878
ICL99	2878
ICL100	2878
ICL101	2879
ICL102	2879
ICL103	2879
ICL104	2880
ICL105	2880
ICL106	2881
ICL107	2881
ICL108	2882
ICL109	2882
ICL110	2882
ICL111	2882
ICL112	2883
ICL113	2883
ICL114	2884
ICL115	2884
ICL116	2884
ICL117	2885
ICL118	2885
ICL119	2885
ICL120	2886
ICL121	2886
ICL122	2886
ICL123	2886
ICL124	2887
ICL125	2887
ICL126	2888
ICL127	2888
ICL128	2888
ICL129	2889
ICL130	2889
ICL131	2890
ICL132	2890
ICL133	2891
ICL134	2891
ICL135	2892
ICL136	2893
ICL137	2893
EDT Rules (K Rules)	2895

Table of Contents

K1	2896
K2	2896
K3	2896
K4	2897
K5	2897
K6	2897
K7	2899
K8	2899
K9	2899
K10	2901
K11	2901
K12	2902
K13	2902
K14	2902
K15	2903
K16	2908
K17	2908
K18	2909
K19	2909
K20	2910
K21	2911
K22	2911
K23	2913
K24	2913
K25	2914
Procedure Rules (P Rules)	2916
P1	2918
P2	2919
P3	2919
P4	2920
P5	2920
P6	2920
P7	2921
P8	2921
P9	2921
P10	2922
P11	2922
P12	2923
P13	2924
P14	2924
P15	2925
P16	2925
P17	2925
P18	2926
P19	2926
P20	2927
P21	2927
P22	2927
P23	2928

Table of Contents

P24	2928
P25	2929
P26	2929
P27	2929
P28	2930
P29	2931
P30	2931
P31	2932
P32	2932
P33	2932
P34	2933
P35	2933
P36	2934
P37	2934
P38	2934
P39	2935
P40	2935
P41	2935
P42	2936
P43	2936
P44	2937
P45	2937
P46	2938
P47	2938
P48	2938
P49	2939
P50	2939
P51	2939
P52	2940
P53	2940
P54	2941
P55	2942
P56	2942
P58	2943
P59	2943
P60	2943
P62	2944
P63	2944
P64	2945
P65	2945
P66	2945
P70	2946
P71	2946
P72	2947
P73	2947
P74	2947
P75	2948
P76	2949
P77	2949

Table of Contents

P78	2950
P79	2950
P80	2951
P81	2952
P82	2952
P83	2953
P84	2953
P85	2954
P86	2954
P87	2955
P88	2955
P89	2956
P90	2956
P91	2957
P92	2957
P93	2958
P94	2958
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)	2961
R1	2961
R2	2962
R3	2962
R4	2963
R5	2963
R6	2964
R7	2964
R8	2965
R9	2968
R10	2969
R11	2970
R12	2971
R13	2972
R14	2972
R15	2974
R16	2977
R17	2979
R18	2979
R19	2980
R20	2980
R22	2981
R23	2982
R27	2982
R28	2983
Scannability Rules (S Rules)	2984
S1	2985
S2	2986
S3	2987
S4	2988
S5	2989
S6	2990

Table of Contents

S7	2991
S8	2992
Scan Chain Trace Rules (T Rules)	2994
T1	2995
T2	2995
T3	2996
T4	3000
T5	3005
T6	3007
T8	3008
T9	3008
T10	3009
T11	3009
T12	3009
T13	3010
T14	3010
T15	3010
T16	3011
T17	3011
T18	3012
T19	3012
T20	3013
T21	3013
T22	3013
T23	3014
T24	3014
T25	3015
T26	3016
Power-Aware Rules (V Rules)	3018
V1	3019
V2	3020
V3	3020
V4	3020
V5	3021
V6	3021
V7	3022
V8	3022
V9	3023
V10	3024
V11	3025
V12	3025
V13	3026
V14	3027
V15	3028
V16	3028
V17	3028
V18	3029
V19	3029
V20	3030

Table of Contents

V21	3030
Timing Rules (W Rules)	3032
W1	3033
W2	3033
W3	3034
W4	3034
W5	3034
W6	3035
W7	3035
W8	3036
W9	3036
W10	3036
W11	3037
W12	3037
W13	3038
W14	3038
W15	3038
W17	3039
W18	3039
W19	3040
W20	3040
W21	3041
W22	3041
W23	3041
W24	3042
W25	3042
W26	3043
W27	3043
W28	3044
W29	3044
W30	3045
W31	3045
W32	3046
W33	3046
W34	3047
W35	3048
W36	3048
W37	3048
W38	3049
W39	3050
W41	3051
Other DRC Messages	3052
Transparent Capture Handling Analysis	3052
Oscillation Limitation	3053
RAM Summary Results and Test Capability	3053

Chapter 8	
Data Models.....	3055
Hierarchical Design Data Model	3056
Module	3058
Instance	3069
Port	3080
Pin	3094
Net	3099
Pseudo_port	3104
Flat Design Data Model.....	3107
Flat Design Object Types	3108
Gate_pin	3108
ICL Data Model.....	3118
ICL Object Types	3119
icl_module	3119
icl_port	3126
icl_instance	3139
icl_pin	3141
Scan Data Model	3144
Scan Element Object Type	3144
Scan Chain Family Object Type.....	3150
Scan Mode Object Type	3151
Test Point Data Model.....	3151
Configuration Data Model.....	3153
config_csv_wrapper	3153
config_data_wrapper	3154
config_property	3155
config_repeatable_property	3155
config_wrapper	3156
Chapter 9	
Attributes.....	3159
Attribute-Related Commands	3159
Attribute Types	3161
Pre-Defined Attributes	3161
User-Defined Attributes	3161
Attribute Value Types	3161
Attribute Filtering Equation Syntax.....	3162
Glob and Regular Expression Pattern Matching Syntax	3164
Hierarchical Name Matching With Escaped Identifiers	3167
Attribute Inheritance Behavior	3170
Inheritance Rules	3170
DFT Test Logic Related Attributes (no_control_point and no_observe_point).	3171
Chapter 10	
Configuration-Based Specification	3175
Interface Naming Limitations	3177
DftSpecification Configuration Syntax	3178

Table of Contents

DftSpecification	3179
EDT	3184
IjtagNetwork	3224
InSystemTest	3319
EmbeddedBoundaryScan	3339
BoundaryScan	3384
LogicBist	3445
LpctType3	3466
MemoryBISR	3478
MemoryBIST	3510
OCC	3539
RtlCells	3563
PatternsSpecification Configuration Syntax	3569
PatternsSpecification	3570
Patterns	3576
SimulationOptions	3580
AdvancedOptions	3585
TestStep	3590
ICLNetworkVerify	3655
ProcedureStep	3659
EnableGroupInfo	3669
SdfInfo	3671
LoadBoardInfo	3674
TesterInterface	3676
Loopbacks	3679
ACLoopbacks	3681
InSystemTest	3683
DefaultsSpecification Configuration Syntax	3685
DefaultsSpecification	3686
DefaultsSpecification/DftSpecification	3687
IjtagNetwork	3689
BoundaryScan	3693
MemoryBISR	3699
MemoryBIST	3703
DefaultsSpecification/PatternsSpecification	3714
SignOffOptions	3715
ManufacturingOptions	3717
Metadata Configuration Syntax	3719
Wrapper	3721
Property	3730
RepeatableProperty	3735
DataWrapper	3738
CsvWrapper	3741
Id	3743
Value	3747
RowIdElement	3751

Chapter 11	
Tessent Core Description	3755
Scan	3756
Chapter 12	
Tessent Shell Data Base (TSDB)	3763
root_directory	3763
instruments	3765
dft_inserted_designs	3768
patterns	3774
logic_test_cores	3776
<design_name>.lbist_mode[<mode_name>]	3777
<design_name>.atpg_mode[<mode_name>]	3778
<design_name>.atpg_retargeting_mode[<mode_name>]	3779
Chapter 13	
Instrument Connectivity Language (ICL)	3781
ICL Overview	3783
Unsupported ICL Language Elements	3783
Common Features of the ICL Language	3784
Numbers	3784
Strings	3785
Identifiers	3785
Limited Identifier Mathematics	3786
Concatenations	3786
ICL Statement Descriptions	3788
Syntax Conventions	3789
Module	3790
Attribute	3792
ScanInterface	3794
Chain	3800
ScanInPort	3803
ShiftEnPort	3805
CaptureEnPort	3807
UpdateEnPort	3809
DataInPort	3811
SelectPort	3814
ResetPort	3816
TMSPort	3818
TCKPort	3821
ClockPort	3823
TRSTPort	3825
AddressPort	3826
WriteEnPort	3828
ReadEnPort	3830
ScanOutPort	3832
DataOutPort	3835
ToShiftEnPort	3838

Table of Contents

ToCaptureEnPort	3842
ToUpdateEnPort	3846
ToResetPort	3850
ToSelectPort	3853
ToClockPort	3856
ToTCKPort	3861
ToTMSPort	3863
ToTRSTPort	3865
ToIRSelectPort	3867
Instance	3869
Parameter	3875
LocalParameter	3878
Enum	3879
Alias	3882
LogicSignal	3885
ScanRegister	3887
DataRegister	3892
ScanMux	3897
DataMux	3899
ClockMux	3901
OneHotDataGroup	3903
OneHotScanGroup	3907
AccessLink	3909
NameSpace	3918
UseNameSpace	3919
Chapter 14	
Adding Custom Plugin Functionality	3921
plugin directory	3921
Chapter 15	
Generation of ScanDEF	3931
ScanDEF File Generation	3931
ScanDEF File Format	3932
Appendix A	
Parameter File Format and Keywords	3937
What is a Parameter File?	3940
Parameter File Syntax	3940
Parameter File Keywords	3942
A Note about the ALL_* Parameter File Keywords	3945
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS	3945
ALL_EXCLUDE_POWER_GROUND	3946
ALL_EXCLUDE_UNUSED	3946
ALL_FIXED_CYCLES	3947
ALL_FLATTEN_TIMING	3948
ALL_FULL_TIMEPLATES	3948
ALL_HEADER_COMMENT	3948

ALL_IDDQ_TESTER_CYCLE	3949
ALL_MAX_LOOP	3950
ALL_MIN_SCAN_LOAD	3951
ALL_NO_CYCLE_OPT	3951
ALL_NO_LOOP	3951
ALL_NO_PATTERN_TYPE	3951
ALL_NONSCAN_CONSTANT	3952
ALL_ORIGINAL_INDEX	3952
ALL_PRESERVE_FORCE_X	3953
ALL_TIME_RESOLUTION	3953
ALL_USE_CYCLE_LOOP	3953
CTL_ONE_PAT	3954
CTL_NO_PROTO	3954
CTL_STIL_0	3954
CTL_STIL_1	3955
FTDL_DESIGNER	3955
FTDL_MAX_PAT_BLOCK	3955
FTDL_REVISION	3956
FTDL_TCOMMENT	3956
FTDL_TEST_NAME	3956
FTDL_TEST_NUM	3957
FTDL_TEST_SUFFIX	3957
FTDL_ZMODE_MES	3957
SIM_ANNOTATE QUIET	3958
SIM_CHAIN_ERROR	3958
SIM_CHANGE_PATH	3959
SIM_COMPARE_SUMMARY	3959
SIM_COMPARE_X	3959
SIM_DELAY_SCAN_RELEASE	3960
SIM_DIAG_FILE	3961
SIM_DISTRIBUTED_PAT	3961
SIM_DUMPFILE_PATH	3962
SIM_EARLY_RELEASE	3962
SIM_EARLY_RELEASE_LAST	3963
SIM_EARLY_RELEASE_TIME	3963
SIM_FILL_BIDISCAN	3964
SIM_FORCE_Z_AT_CYCLE	3964
SIM_INCLUDE	3964
SIM_INSTANCE_NAME	3965
SIM_KEEP_PATH	3965
SIM_MASK_FILE	3966
SIM_MIN_NAME_FILE	3966
SIM_MISCOMPARE_LIMIT	3967
SIM_MODULE_INCLUDE	3967
SIM_MODULE_NAME	3967
SIM_NAME_FILE	3968
SIM_NO_CHAINNAME_FILE	3968
SIM_NO_INTERNAL_COMPARE	3969
SIM_NO_MEASURE_IN	3969

Table of Contents

SIM_PARALLEL_DELAY_SHIFT_CLOCKS	3969
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE.....	3970
SIM_POST_SHIFT	3970
SIM_PRE_SHIFT.....	3971
SIM_PRECISION.....	3971
SIM_SERIAL_CHAIN.....	3971
SIM_SHIFT_DEBUG	3972
SIM_SIGNAL_SPY	3973
SIM_STATUS_MSG	3973
SIM_STRETCH_SCAN_RELEASE	3973
SIM_TIMEPLATE_COMM	3974
SIM_TMP_REG_LENGTH.....	3974
SIM_TOP_NAME	3974
SIM_VECTOR_COMM.....	3975
SIM_VECTYPE_SIGNAL.....	3976
STIL_2005_SCAN	3977
STIL_CHAIN_CELLS	3978
STIL_CHANNEL_CELLS.....	3978
STIL_CLOCK_WFC_PULSE	3979
STIL_DUAL_MODE	3979
STIL_FULL_CHAIN	3980
STIL_IDDQ_UPPERCASE	3980
STIL_MIN_QUOTE.....	3980
STIL_NESTED_MACRO	3981
STIL_NO_SCANSTRUCTURE	3981
STIL_NO_SCANX	3981
STIL_NOMEASURE_CLOCK	3982
STIL_PAT_ANN	3982
STIL_PAT_CMT	3983
STIL_PAT_LAB	3983
STIL_QUAD_MODE	3983
STIL_QUOTE_ALL	3984
STIL_SCAN_ANN1	3984
STIL_STRUCTURAL	3985
STIL_TI_TITLE	3987
STIL_TRIM_PULSE	3988
STIL_VECTOR_ANN	3988
STIL_VERG_ESC	3990
STIL_WRAP_ANNOTATION	3990
TITDL_CHAIN_SETTYPE	3990
TITDL_CUSTOMER	3991
TITDL_GROUP_TIMING	3991
TITDL_KEEP_SCANOUT	3991
TITDL_LIBRARY	3992
TITDL_PARTNUM	3992
TITDL_PSEUDO_PREFIX	3992
TITDL_REVISION	3993
TITDL_SET_DESCRIPTION	3994
TITDL_SETNAME	3994

TITDL_SETTYPE	3994
WGL_ADD_LASTX	3995
WGL_ALT_BIDI	3995
WGL_ALT_VECT_ANN	3996
WGL_EDGE_STROBE	3996
WGL_FULL_CHAIN	3997
WGL_FULL_SCANGROUP	3998
WGL_GROUP_PIN	3999
WGL_INV_SC	3999
WGL_NO_SCANX	4000
WGL_NOMEASURE_CLOCK	4000
WGL_ONE_ILLINOIS	4001
WGL_PATTERN_NAME	4001
WGL_TRIM.LEADING_P	4001
WGL_TRIM.PULSE	4002
WGL_VECTOR_ANN	4002
WGL_VERG_ESC	4003
WGL_VTRAN_PADSC	4003
WGL_WRAP_ANNOTATION	4003

Appendix B**HDL Limitations in the Tessent Shell Flow** **4005**

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog	4007
HDL Limitations in the Tessent Shell Flow for VHDL	4010
SystemVerilog Interfaces	4013
Multi-dimensional Instantiation of SystemVerilog Interfaces	4013
SystemVerilog Interfaces Cannot Be Edited	4013
Outside Connections on Interface Type Ports Cannot be Intercepted	4013
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type	4014
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block	4014
System Verilog Interfaces in Task or Function Invocation	4015
Nested Modules	4015
SystemVerilog Enumerations	4017
SystemVerilog Enum Methods	4017
SystemVerilog Enum Type in Port Declarations	4017
Complex Port Types for Verilog and SystemVerilog	4018
Access to Ports and Nets of Complex Types	4018
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design	4019
Complex Types in Top-Level Ports at the Chip Level	4019
Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog	4020
SystemVerilog Extern Modules	4020
Unrolled Generate Loops for Verilog and SystemVerilog	4021
Unrolling of Generate Loops Side Effects	4021
Insertion of Custom Logic Which Connects to a Port or Net in an Unrolled Generate Loop	4024
Parameterized Designs for Verilog and SystemVerilog	4025
Parameter Overrides in the Instantiation of the Root Module of a Child Block	4025

Table of Contents

Support for the Defparam Construct	4025
Implicit Port Syntax	4026
SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block	
4026	
SystemVerilog 2009	4027
New Module Insertion in VHDL	4028
Insertion of VHDL Entities with Unconstrained Ports	4028
Insertion of VHDL Entities with Uninitialized Generics	4028
Complex Port Types for VHDL	4029
Access to Ports and Nets of Complex Types	4029
Array of Array and Record Types in Port Declarations in the Root Module of a Design .	4029
Complex Types in Top-Level Ports at the Chip Level	4029
Ports with Unconstrained Array Types in the Root Design Entity of a Child Block . . .	4030
Procedures, Functions, and Process Statements for VHDL	4030
Configurations for VHDL	4030
VHDL 2008	4030
Parameterized Designs for VHDL	4032
Generic Overrides in the Instantiation of the Root Design Entity of a Child Block . . .	4032

Appendix C Getting Help **4033**

The Tessent Documentation System	4033
Mentor Support Services	4034

Index

Third-Party Information

End-User License Agreement

List of Figures

Figure 3-1. Illustration of various clock types.....	129
Figure 3-2. Merge Patterns Sets into One Patterns Set	154
Figure 3-3. Merging Patterns for Top Level and Core at the Top Level.....	155
Figure 3-4. Example Usage of add_dft_clock_mux	168
Figure 3-5. Example Usage of add_dft_control_points Command	175
Figure 3-6. Multiplexing and Pipelining Logic for Output Connections	178
Figure 3-7. Enable Logic and Pipelining for Input Connections.....	179
Figure 3-8. Multiplexing for Destinations of Modal Connections	179
Figure 3-9. Timing Arcs for Scan In and Out Paths	180
Figure 3-10. Waveforms of Selected Dynamic DFT Signals	212
Figure 3-11. add_display_callout Example Result	224
Figure 3-12. Example of a False Path to be Defined.....	250
Figure 3-13. Example Circuit for Defining False Paths	251
Figure 3-14. Static Fault	263
Figure 3-15. Transition Fault - Incompatible Clocks	264
Figure 3-16. Transition Fault - Compatible Clocks.....	265
Figure 3-17. Transition Fault - No Equivalent Pins.....	265
Figure 3-18. Transition Fault - Equivalent Pins	266
Figure 3-19. Example Scan Interfaces.....	271
Figure 3-20. Logical Connection Attributes	283
Figure 3-21. Logical Connection Example	284
Figure 3-22. MISR placement	295
Figure 3-23. analyze_simulation_mismatches Example	416
Figure 3-24. Extra Layout Hierarchy Example	510
Figure 3-25. Extra Design Hierarchy Example	511
Figure 3-26. Hierarchical Test Logic Insertion Example	653
Figure 4-1. Selecting Two Nodes in the Configuration Tree	783
Figure 4-2. Various Components of a Configuration Element	785
Figure 4-3. Circuit Illustrating the Usage of the get_fanins Command.....	847
Figure 4-4. Circuit Illustrating the Usage of the get_fanouts Command.....	850
Figure 4-5. iApply Time Frame	1046
Figure 4-6. Inverter Interception	1094
Figure 4-7. Intercepting an Input Pin From Inside An Instance	1148
Figure 4-8. Intercepting an Output Port From Inside the Current Design.....	1149
Figure 4-9. Intercepting an Input Pin From Outside An Instance	1150
Figure 4-10. Intercepting an Output Pin From Outside An Instance.....	1151
Figure 4-11. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 0	1160
Figure 4-12. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 0	1161
Figure 4-13. Scan Frame Timing with tck_ratio = 4 and TCK Off Value of 0	1161
Figure 4-14. Scan Frame Timing with tck_ratio >= 8 and TCK Off Value of 0	1162

Figure 4-15. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 1	1162
Figure 4-16. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 1	1163
Figure 4-17. Scan Frame timing with tck_ratio = 4 and TCK Off Value of 1	1163
Figure 4-18. Scan Frame timing with tck_ratio >= 8 and TCK Off Value of 1	1164
Figure 5-1. Design Views in a Bottom-up Flow	1216
Figure 5-2. TSDB logic_test_cores Directory Structure	1268
Figure 5-3. Case Analysis Example	1277
Figure 5-4. Plugin Directory Contains XYZ Custom DRC Rules	1321
Figure 5-5. Violation Detected by XYZ_S1 DRC	1322
Figure 5-6. Combinational Paths Allowed and Rejects by XYZ_S2	1336
Figure 5-7. Violation Detected by XYZ_S3	1339
Figure 5-8. Forward Walk to Find All Destination Hits	1342
Figure 5-9. Backward Walk to Find All Source Hits	1342
Figure 5-10. Violations Detected by XYZ_S4_DRC	1344
Figure 5-11. set_clock_off_simulation Off and set_split_capture_cycle Off	1819
Figure 5-12. set_clock_off_simulation On and set_split_capture_cycle Off	1819
Figure 5-13. set_clock_off_simulation Off and set_split_capture_cycle On	1819
Figure 5-14. set_clock_off_simulation On and Set Split Capture On	1820
Figure 5-15. Typical Contents of the synthesis_outdir Directory	1863
Figure 5-16. Example Output Directory Structure (Questa)	1872
Figure 5-17. Outdir Directory Structure for VCS	1875
Figure 5-18. Output Directory Structure for Incisive	1876
Figure 6-1. Global versus Local Bus Simulation Analysis Example	1940
Figure 6-2. Half-cycle Skew	1985
Figure 6-3. One-cycle Skew	1986
Figure 6-4. Multiple-cycle Skew	1986
Figure 6-5. False Path Intersection and Effect Cones	2154
Figure 6-6. Example of a Reconvergent Controlling Path	2488
Figure 6-7. Flat Schematic Window After add_display_instance Command	2492
Figure 6-8. Result of trace_flat_model While Tagging Multiplexers	2493
Figure 6-9. Writing Out the Modified Design in rtl vs no_rtl Modes	2530
Figure 6-10. Effect of Successive Design Editing Passes on the Design View	2531
Figure 7-1. Example of Design Level	2647
Figure 7-2. Example of Primitive Level	2647
Figure 7-3. Data Reported for a Specific Gate	2649
Figure 7-4. Clock Cycle Terminology	2688
Figure 7-5. Example Clock Cone	2689
Figure 7-6. Example Effect Cone	2689
Figure 7-7. Pin in Both Cones	2690
Figure 7-8. DFTVisualizer Flat Schematic Window View Showing Clock Cones	2690
Figure 7-9. Clock Source And Derived Clocks	2691
Figure 7-10. C1 Violation Example	2694
Figure 7-11. C2 Rule Example Circuit	2698
Figure 7-12. C3 Violation Example	2699
Figure 7-13. Example Timing That Allows Sink to Capture Source's New Data	2700

List of Figures

Figure 7-14. C4 Violation Example	2706
Figure 7-15. Example Where Actual Behavior Differs from Tool's Prediction	2707
Figure 7-16. C5 Rule Example Circuit	2713
Figure 7-17. C6 Rule Example Circuit	2715
Figure 7-18. C7 Rule Example Circuit	2717
Figure 7-19. C8 Rule Example Analysis	2718
Figure 7-20. C9 Rule DRC Analysis	2720
Figure 7-21. C10 Rule Example Circuit	2721
Figure 7-22. C22 DRC Example Design 1	2727
Figure 7-23. C22 DRC Example Design 2	2728
Figure 7-24. Rule D10 Violation Example	2745
Figure 7-25. Example DFT_C1 violation in the presence of a blackbox	2752
Figure 7-26. Example DFT_C1 Violation on Differential Clocks	2754
Figure 7-27. Schematic and Flat Model View of a Differential Clock Path	2754
Figure 7-28. Example of DFT_C4 Violation	2757
Figure 7-29. Example DFT_C6 Violation	2759
Figure 7-30. DFT_C9 example of type 1 violation	2763
Figure 7-31. Race Condition Check	2764
Figure 7-32. Example E4 Violation Trace in Flat Schematic Window of DFTVisualizer . .	2779
Figure 7-33. Example E10 Violation Trace in Flat Schematic Window of DFTVisualizer .	2787
Figure 7-34. Blocked Decompressor Input	2798
Figure 7-35. Blocked Channel Input to Decompressor	2799
Figure 7-36. Output Channel and Compactor Connection Blocked	2801
Figure 7-37. Viewing F8 DRC Violation in DFTVisualizer	2803
Figure 7-38. Tracing Back on the F8 X path	2804
Figure 7-39. Tri-State Driven to X by connection to VDD	2804
Figure 7-40. F8 DRC Violation - Blocked	2805
Figure 7-41. F8 DRC Violation Cause	2806
Figure 7-42. Blocked Scan Chain Path	2806
Figure 7-43. analyze_drcViolation F22 in Test Structures Window	2819
Figure 7-44. Callout Text for reg_lfsm_vec_lockup_3	2819
Figure 7-45. analyze_drcViolation F22 in Flat Schematic Window	2819
Figure 7-46. Invalid multi-TAP Configuration	2892
Figure 7-47. Valid multi-TAP Configuration	2892
Figure 7-48. Failing identification	2900
Figure 7-49. Common cell structure	2900
Figure 7-50. Failing identification	2901
Figure 7-51. Example S3 Rule Violation	2988
Figure 7-52. T3 Violation	2997
Figure 7-53. Tracing T3 Violation	2998
Figure 7-54. T4 Violation	3002
Figure 7-55. T4 Violation Traced Back	3002
Figure 7-56. T4 Violation Showing States	3003
Figure 7-57. T4 Violation Traced Back Showing States	3003
Figure 7-58. T5 DRC Violation	3006

Figure 7-59. T5 DRC Violation Traced Back	3006
Figure 7-60. Example of T25 Violation	3015
Figure 7-61. W38 Reference Circuit	3050
Figure 7-62. Transparent Capture Handling Analysis Messages	3052
Figure 7-63. RAM Summary Results and Test Capability Messages	3054
Figure 8-1. Tessent Shell Hierarchical Design Objects	3056
Figure 8-2. Tessent Shell Non-Hierarchical Design Objects	3057
Figure 9-1. Hierarchical Test Logic Insertion Example	3172
Figure 10-1. IJTAG Network Example Schematic	3227
Figure 10-2. Existing Compliance Enable Modules	3232
Figure 10-3. DataInPorts Example Schematic	3237
Figure 10-4. HostScanInterface/Interface Example 1 Schematic	3246
Figure 10-5. HostScanInterface/Interface Example 3 Schematic	3249
Figure 10-6. DesignInstance Example Schematic	3253
Figure 10-7. ScanMux Internal Schematic When Used on a DR Scan Path	3255
Figure 10-8. ScanMux Internal Schematic When Used on a TAP Scan Path	3256
Figure 10-9. ScanMux Example Schematic	3259
Figure 10-10. SIB Internal Schematic	3265
Figure 10-11. Sib(sti) With Scan Isolation Chain	3267
Figure 10-12. Added Scan Isolation Logic for SIB(sti)	3268
Figure 10-13. Mini-OCC for Scan Tested TCK Domain	3269
Figure 10-14. SIB Example Schematic	3272
Figure 10-15. Update Stage Timing Diagram	3278
Figure 10-16. Detailed View of TDR Schematic	3281
Figure 10-17. DataOutPort Connection Multiplexing	3290
Figure 10-18. TAP FMS Diagram	3296
Figure 10-19. TAP Block Diagram	3297
Figure 10-20. Example Multi-TAP Configuration Schematic	3299
Figure 10-21. Example of TDR Scan Selector	3300
Figure 10-22. Example Design With Pads	3359
Figure 10-23. Single Logical Group Example	3360
Figure 10-24. Two Logical Groups Example	3360
Figure 10-25. Two Logical Groups Example1 with One Output per Enable Cells	3361
Figure 10-26. Two Logical Groups Example2 with One Output per Enable Cells	3361
Figure 10-27. Typical Boundary Scan Cell Inserted for an Inout Pad	3366
Figure 10-28. Boundary Scan Cell as Observation and Control in Logic Test Modes	3367
Figure 10-29. Adding Auxiliary Input and Output Logic in Bscan Cell	3368
Figure 10-30. Path Used When AuxOut Logic Activated	3369
Figure 10-31. Forcing aux_out_en Low in Capture Mode to Observe Core Output	3370
Figure 10-32. Using AuxIn and Controlling to Core Signal in ATPG	3371
Figure 10-33. Extra Logic Added When a Port is Used Both Internally and Externally	3372
Figure 10-34. Needed Resources From Third-Party TAP Controller	3385
Figure 10-35. Schematic of the Boundary Scan Interface Module	3391
Figure 10-36. Segmenting the Boundary Scan Chains into Logic Test Chains	3392
Figure 10-37. Example pin_order_file	3396

List of Figures

Figure 10-38. Example Design With Pads	3413
Figure 10-39. Single Logical Group Example	3414
Figure 10-40. Two Logical Groups Example	3414
Figure 10-41. Two Logical Groups Example1 with One Output per Enable Cells	3415
Figure 10-42. Two Logical Groups Example2 with One Output per Enable Cells	3415
Figure 10-43. Typical Boundary Scan Cell Inserted for an Inout Pad	3420
Figure 10-44. Boundary Scan Cell as Observation and Control in Logic Test Modes	3421
Figure 10-45. Adding Auxiliary Input and Output Logic in Bscan Cell	3422
Figure 10-46. Path Used When AuxOut Logic Activated	3423
Figure 10-47. Forcing aux_out_en Low in Capture Mode to Observe Core Output	3424
Figure 10-48. Using AuxIn and Controlling to Core Signal in ATPG	3425
Figure 10-49. Extra Logic Added When a Port is Used Both Internally and Externally	3426
Figure 10-50. package0 Bonding Configuration Example	3441
Figure 10-51. package1 Bounding Configuration Example	3442
Figure 10-52. package2 Bonding Configuration Example	3442
Figure 10-53. BISR Segment Order File	3480
Figure 10-54. Case 1 — clock_intercept_node Has Functional Source and fast_clock Not Specified	3559
Figure 10-55. No Functional Source and fast_clock Specified	3559
Figure 10-56. Case 3 - Functional Source and fast_clock Specified	3560
Figure 10-57. Timing Diagram of the Global tristate_enable_non_contacted Test	3604
Figure 10-58. Timing Diagram of the Local tristate_enable_non_contacted Test	3605
Figure 10-59. Fast BISR Muxing	3622
Figure 10-60. Example Chip with Two Cores and Complex Clocking	3668
Figure 10-61. Example Option panel where blank spaces were added	3732
Figure 12-1. Root Directory Structure	3764
Figure 15-1. Single Scan Mode Environment	3932
Figure 15-2. Multi Scan Mode Environment	3932

List of Tables

Table 2-1. Conventions for Command Line Syntax	68
Table 2-2. Tool Invocation Command Summary	68
Table 3-1. Conventions for Command Line Syntax	81
Table 3-2. Commands A Through D	82
Table 3-3. Example Cell Value Changes with Different Constraints for Scan Patterns	119
Table 3-4. Example Scan Cell Value Changes with Different Constraints for Chain Patterns	
120	
Table 3-5. Core Instance Parameters and Values by Instrument	149
Table 3-6. Mapping of Core Descriptions and Instances	156
Table 3-7. Mapping of Core Descriptions With Modules and Instances	157
Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations	160
Table 3-9. Dft Logic for add_dft_control_points Nodes	170
Table 3-10. Pre-Registered Static DFT Signal Names	190
Table 3-11. Pre-Registered Dynamic DFT Signal Names	209
Table 3-12. How the DFTVisualizer Flat Schematic Window Displays Instances	228
Table 3-13. ICL Port Types	268
Table 3-14. Logical Connection Attributes	282
Table 3-15. Fault Class Codes and Class Names	608
Table 4-1. Conventions for Command Line Syntax	705
Table 4-2. Commands E Through Q	706
Table 4-3. Types of Configuration Elements	786
Table 4-4. Available Partitions for Introspection	790
Table 4-5. ICI objects and which options they apply to	873
Table 4-6. Source Types	893
Table 4-7. Events of type “vector”	957
Table 4-8. Events of type “scan”	958
Table 4-9. Events of type “reset”	960
Table 4-10. Events of type “loop”	961
Table 4-11. Events of type “goto_state”	961
Table 4-12. Events of type “nop”	961
Table 4-13. Interception based on node type	1089
Table 4-14. Allowed Cell Function Names	1091
Table 4-15. RunLoop Scaling Algorithm	1113
Table 5-1. Conventions for Command Line Syntax	1187
Table 5-2. Commands R	1188
Table 5-3. Mask File Statements	1263
Table 5-4. Named Capture Procedure Merging Under Different Conditions	1271
Table 5-5. Mode values for the RTL Synthesis Proc	1318
Table 5-6. Clock Compatibility Summary	1425
Table 5-7. Available Partition for Introspection	1440

Table 5-8. Default Netlist Information Reported for Instances	1482
Table 5-9. Available Information Displayed and Arguments	1485
Table 5-10. Fault Class Codes and Names	1557
Table 5-11. Name Conventions Used by report_faults -Cell_name	1560
Table 5-12. Reportable Gate Types	1587
Table 5-13. Clock Port Categories	1589
Table 5-14. Notation Used in report_patterns Display	1701
Table 5-15. Default report_processors Information Display	1725
Table 5-16. Additional Data Displayed by -Verbose	1726
Table 5-17. Format Codes	1788
Table 5-18. Reason for Failed Identification	1838
Table 5-19. Untestable Faults that are Reclassified by reset_au_faults	1848
Table 6-1. Conventions for Command Line Syntax	1887
Table 6-2. Commands S Through Z	1888
Table 6-3. User-Specifiable Attributes	1919
Table 6-4. Default Clock Compatibility Summary at a Clock Domain Boundary	1984
Table 6-5. Available Partition for Introspection	1994
Table 6-6. Contexts Available by License	2004
Table 6-7. Predefined Simulation Contexts	2023
Table 6-8. Default Extensions for set_design_sources	2039
Table 6-9. Pattern Sources	2141
Table 6-10. Allowed TAP States	2188
Table 6-11. Annotation Parameter Values	2190
Table 6-12. Pin Functions That Can Be Renamed	2214
Table 6-13. WIRE Bus Contention Truth Table	2275
Table 6-14. AND Bus Contention Truth Table	2275
Table 6-15. OR Bus Contention Truth Table	2276
Table 6-16. Keywords for Pattern Types	2292
Table 6-17. Mask File Statements	2305
Table 6-18. Fault Classes for Relevant Coverage	2346
Table 6-19. Fault Sub-Classes for Relevant Coverage	2347
Table 6-20. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch	2426
Table 6-21. X Handling for Level-Sensitive Ports	2464
Table 6-22. X Handling for Edge-Triggered Ports	2465
Table 6-23. Difference between rtl and Netlist files	2530
Table 6-24. Single Module Per File Naming Convention	2546
Table 6-25. Name Conventions Used by write_faults -Cell_name	2573
Table 7-1. DRC Violation Messages	2642
Table 7-2. Clocking that Can Result in a C3 Signal Race	2698
Table 7-3. Clocking that Can Result in a C4 Signal Race	2705
Table 7-4. Error Messages Generated by the Clock Tracing Checks	2752
Table 7-5. Error Messages Generated by Differential Clock Checks	2753
Table 7-6. Error Messages Generated by the Clock Tracing Checks	2759
Table 7-7. Error messages Generated by DFT_C7 Checks	2760
Table 7-8. Error messages Generated by DFT_C8 Checks	2762

List of Tables

Table 7-9. Error messages Generated by DFT_C9 Checks	2762
Table 7-10. Error and Warning Messages Generated by DFT_C10	2766
Table 7-11. Warning Messages Generated by DFT_C11 Checks	2772
Table 7-12. ICL Extraction DRCs	2834
Table 7-13. Port Connections	2837
Table 7-14. Port Types and Required Input Binding	2861
Table 8-1. Built-in Primitives and Primitive Port Names	3113
Table 8-2. Built in attributes on Scan Element (SE) Object Type	3145
Table 8-3. Built in Attributes on Scan Chain Family Object Type	3150
Table 8-4. Built-in Attributes on Scan Mode Object Type	3151
Table 8-5. Built in Attributes on Test Point Object Type	3152
Table 9-1. Commands for Manipulating Attributes	3159
Table 9-2. Attribute Value Types	3161
Table 9-3. Posix Extended Regular Expression Metacharacters	3164
Table 9-4. Special Characters in Regular Expressions	3166
Table 9-5. Special Characters Set in Regular Expression	3166
Table 10-1. Configuration Data Editing and Introspection Commands	3175
Table 10-2. SyntaxConventions for Configuration Files	3176
Table 10-3. Illustration of Different Boundary Scan Implementations	3346
Table 10-4. BoundaryScanCellOptions list	3350
Table 10-5. Illustration of the “sample_only” Boundary Scan Cell Option	3354
Table 10-6. Connections to and from Internal Boundary Scan Cells	3380
Table 10-7. Illustration of Different Boundary Scan Implementations	3400
Table 10-8. BoundaryScanCellOptions list	3404
Table 10-9. Illustration of the “sample_only” Boundary Scan Cell Option	3408
Table 10-10. Connections to and from Internal Boundary Scan Cells	3434
Table 10-11. ROM MISR Signature Source	3651
Table 10-12. DramOptions Parameter Values	3653
Table 12-1. Instrument Container Names	3765
Table 12-2. Format to extension mapping for the interface view	3771
Table 13-1. SyntaxConventions for ICL Files	3789
Table A-1. Vector Types	3975
Table A-2. Translation Table	3976
Table A-3. STIL Special Keywords	3989
Table A-4. STIL Pattern Types	3989
Table A-5. STIL Vector Types	3989
Table B-1. Verilog and SystemVerilog Limitations	4007
Table B-2. VHDL Limitations	4010

Chapter 1

Introduction

Tessent Shell is an environment that provides access to all Tessent products.

Overview.....	53
Product Introduction.....	56
Tessent BoundaryScan	56
Tessent Diagnosis	58
Tessent FastScan and Tessent TestKompress.....	58
Tessent IJTAG	61
Tessent LogicBIST	61
Tessent MemoryBIST	62
Tessent MissionMode	63
Tessent Scan	63
Tessent ScanPro	64
Tessent SiliconInsight.....	65
Tessent DesignEditor	65

Overview

Application commands are provided for Tessent Shell, as well as for the following products:

- Tessent BoundaryScan is a complete solution for the creation and integration of boundary scan cells and related control logic for embedded test and diagnosis of integrated circuit I/Os, as well as test and diagnosis of board-level interconnects between ICs. Tessent BoundaryScan provides a completely automated solution for adding standard boundary scan support to ICs of any size or complexity, reducing IC engineering development effort and improving time-to-market.
- Tessent Diagnosis is a sophisticated diagnosis tool used at different stages of test to help characterize and locate defects in manufactured chips. It is for users who need to shorten the time it takes to perform yield ramp, yield learning and customer return analysis. The unique advanced diagnosis capabilities it provides are also available in Tessent TestKompress.
- Tessent FastScan is a comprehensive combinational Automatic Test Pattern Generation (ATPG) system optimized for full-scan designs. It offers high speed and accurately measured high test coverage to guarantee your product quality and reliability.
- Tessent IJTAG is Mentor Graphics implementation of the [IEEE 1687-2014](#) (IJTAG) and supports reading Instrument Connectivity Language (ICL), performing automatic retargeting of IJTAG-specific PDL commands such as iWrite and iRead, and writing out

the retargeted PDL patterns in many different ATE tester specific formats and as Verilog HDL simulation test benches. See the [Tessent IJTAG User's Manual](#) for complete information.

- Tessent LogicBIST contains all of the functionality needed to implement the Hybrid TK/LBIST Flow for your design. The tool allows you to share the logic between EDT and LBIST controllers that can be configured to work either in EDT or LBIST mode. The product allows you to create a BIST-ready design, generates the hybrid IP, performs fault simulation, and creates top-level serial patterns to validate the LBIST mode.
- Tessent MemoryBIST provides a complete solution for at-speed testing, diagnosis, repair, debug, and characterization of embedded memories. The solution's architecture is hierarchical, allowing BIST and self-repair capabilities to be added to individual cores as well as at the top level.
- Tessent MissionMode provides a combination of automation and on-chip IP for enabling semiconductor chips throughout an automotive (or other safety-critical) electronics system to be tested and diagnosed at any point during the system's functional operation.
- Tessent SiliconInsight provides interactive capabilities for testing, debugging, and characterizing BIST-tested memories and logic, IEEE 1687 IJTAG instruments, and ATPG patterns. It also facilitates on-tester data collection and diagnosis for BIST-tested memories and logic.
- Tessent DesignEditor is a less expensive product that provides access only to design introspection and editing features. This product only provides access to the dft context (commands labeled "dft" or "all contexts").
- Tessent Scan is an internal scan synthesis tool that identifies and inserts scan and test circuitry into your design. For more information, refer to the [Tessent Scan and ATPG User's Manual](#).
- Tessent ScanPro is a tool for inserting test logic to make the design scan and BIST ready. It includes the scan stitching capabilities that are part of Tessent Scan, but in addition includes capabilities for inserting test points to improve pattern counts and test coverage, x-bounding to prevent X's from propagating in the design, and inserts logic to isolate false and multicycle paths.
- Tessent TestKompress uses patented scan EDT™® (Embedded Deterministic Test) technology and many of the industry leading ATPG features of Tessent FastScan, to reduce scan data volume and test time by many times over Tessent FastScan ATPG. It is for users who need greater reductions in ATE memory and time requirements than are available with any other full-scan ATPG tool.
- DFTVisualizer, which is built into all of these tools, is a schematic and data viewing utility. DFTVisualizer adds the ability to graphically investigate and interact with designs, making it easier to debug problems. For more information, see "[DFTVisualizer](#)" in the [Tessent Shell User's Manual](#).

In Tessian Shell, the term “context” refers to a broad category of functionality that often corresponds to a specific product and a license feature, such as Tessian FastScan. By setting the context, you indicate the type of task you want Tessian Shell to perform. For more information about Tessian Shell contexts, refer to the “[Context and System Modes](#)” section in the *Tessian Shell User’s Manual*.

Note

 Design editing commands are not available when using product licenses for Tessian FastScan and Tessian Scan. For more information about the design editing commands, refer to “[Design Editing](#)” in the *Tessian Shell User’s Manual*.

Product Introduction

Tessent Shell is a Tcl shell environment and design data model that provides a unified Tcl command set and command naming conventions.

Tessent Shell also provides access to the following products:

Tessent BoundaryScan	56
Tessent Diagnosis	58
Tessent FastScan and Tessent TestKompress	58
Tessent IJTAG	61
Tessent LogicBIST	61
Tessent MemoryBIST	62
Tessent MissionMode	63
Tessent Scan	63
Tessent ScanPro	64
Tessent SiliconInsight	65
Tessent DesignEditor	65

Tessent BoundaryScan

Tessent BoundaryScan is a complete solution for the creation and integration of boundary scan cells and related control logic for embedded test and diagnosis of integrated circuit I/Os, as well as test and diagnosis of board-level interconnects between ICs. Tessent BoundaryScan provides a completely automated solution for adding standard boundary scan support to ICs of any size or complexity, reducing IC engineering development effort and improving time-to-market.

Tessent BoundaryScan supports standard 1149.1 boundary scan cells, 1149.1 custom boundary scan cells, and optionally 1149.6 boundary scan cells for differential I/O cells driving AC-coupled nets. It also provides a unique 1149.1-based solution for contactless testing of I/Os. For complete information, refer to the *Tessent BoundaryScan User's Manual*.

Tessent BoundaryScan Inputs

To use this flow, you must have either an RTL or a gate-level netlist with IO pads already inserted into the design. For an RTL netlist, you must have the Tessent cell library or the pad library for the pad cells. For a gate-level netlist, you must have the Tessent cell library or the ATPG library for the standard cells, in addition to the Tessent cell library for the IO pad cells. The pad library supported by Tessent BoundaryScan-LV is natively supported in Tessent Shell if the Tessent cell library for the IO pad cells is not present

Tessent BoundaryScan Outputs

The tool creates and interconnects RTL-level boundary scan logic compliant with the IEEE 1149.1-2001 standard.

- **Instruction Support** — Full support of required IEEE standards 1149.1 instructions.
- **Extension Support** — Support of base extensions to IEEE 1149.1, such as the Device ID register.
- **Compliant Verilog** — Generation of Verilog (IEEE 1364-2001) that is compliant with the ModelSim (Verilog), Synopsys' Design Compiler, and other industry synthesis tools.
- **RTL-Level Boundary Scan Generation** — Insertion and interconnection of boundary scan circuitry at the RTL level, moving generation of test circuitry to earlier in the design process.
- **Customized Boundary Scan** — Generation of default or user-customized boundary scan architectures.
- **I/O pad Synthesis** — Generation of generic I/O pads or technology-specific I/O pads.
- **Automatic Connection** — Automatic connection of boundary scan to internal scan logic.
- **Test Bench Generation** — Generation of a test bench, which allows testing of the boundary scan logic after interconnection with the core application logic.
- **Test Vector Generation** — Generation of boundary scan test vectors in a variety of ASIC vendor test data formats, as well as ASCII, binary, STIL, WGL and other pattern formats.
- **Setup File Generation** — Generation of ATPG setup files, for designs with generated boundary scan circuitry controlling internal scan circuitry.
- **Compliant BSDL** — Production of BSDL output that is compliant with IEEE standard IEEE 1149.1-2001 specification.
- **Generic Element Mapping** — Mapping of boundary scan elements to generic boundary scan library cells (which enhances re-targetability of the boundary scan circuitry).
- **Technology-Specific Element Mapping** — Mapping of boundary scan, I/O pad, and TAP controller elements to technology-specific library cells. Technology mapping provides support for replacing generic boundary scan cells, I/O pads, and the TAP controller by equivalent technology-specific cells.

Tessent Diagnosis

Tessent Diagnosis-specific features include many capabilities.

Some of these capabilities are:

- Fault model independent analysis for suspect identification, suspect scoring, and classification.
- Logic and chain diagnosis.
- Direct diagnosis with compressed patterns.
- Links to Calibre® for physical view.

Tessent Diagnosis Inputs

Tessent Diagnosis has the following inputs:

- **Design** — This must be a flattened design, created previously using the [write_flat_model](#) command in Tessent FastScan or Tessent TestKompress.
- **Test Patterns** — STIL, WGL, binary or ASCII version of the patterns applied on the tester.
- **Failure file** — This file contains ATE failure data presented in the correct Mentor Graphics failure file format for the [diagnose_failures](#) command.

Tessent Diagnosis Outputs

Tessent Diagnosis produces the following outputs:

- **Failure Diagnosis Report** — This is a human-readable diagnosis report that is displayed onscreen or written in ASCII format to a file. Additionally, the report can be written to a file in Comma Separated Value (CSV) format.

Tessent FastScan and Tessent TestKompress

Tessent FastScan and Tessent TestKompress share a great deal of functionality. Tessent TestKompress contains the entire Tessent FastScan feature set (the “patterns -scan” context), and in addition provides functionality for EDT IP creation (the “dft -edt” context).

Both products share the following features:

- Can be used within a Mentor Graphics flow or as a point tool in other design flows.
- Contain an internal high-speed fault simulator.
- Produce a number of standard test pattern data formats.
- Contain a powerful design rules checker.

- Read most standard gate-level netlists.
- Produce very high coverage test pattern sets for full-scan and scan-sequential designs. Scan-sequential designs contain well-behaved sequential scan circuitry, including non-scan latches, sequential memories, and limited sequential depth.
- Contain functionality for handling embedded RAM and ROM.
- Support the same pattern types and fault models.
- Provide the same test coverage.
- Provide the same diagnostics.
- Contain the same command set, with the exception that Tessent TestKompress contains additional commands for EDT IP creation and use.

In addition to the previous features, Tessent TestKompress includes the following:

- Increases production throughput: scan test time is much shorter than any other scan methodology.

Tessent FastScan and Tessent TestKompress Inputs

Tessent FastScan and Tessent TestKompress have the following inputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.
- **Test Procedure File** — This file defines the operation of the scan circuitry in your design. You can generate the file by hand or by using the `write_atpg_setup` command in Tessent Scan.
- **Library** — This file contains model descriptions for all library cells used in your design.
- **Test Patterns** — This is a set of externally generated test patterns that you can use as the pattern source for simulation.

Tessent TestKompress has the following inputs:

- EDT IP Creation Phase:
 - **Design** — This must be a gate-level netlist with scan. The supported netlist format is Verilog.
 - **Test Procedure File** — This file defines the operation of the scan circuitry in your design. You can generate the file by hand or by using the `write_atpg_setup` command in Tessent Scan.
 - **Library** — This file contains model descriptions for all library cells used in your design.

- EDT Pattern Generation Phase:
 - **Design** — This must be a scan-inserted, gate-level netlist with EDT circuitry. The supported netlist format is Verilog.
 - **EDT Dofile** — Created by Tessent TestKompress in the IP Creation Phase, this file contains setup commands for the Pattern Generation Phase.
 - **EDT Test procedure file** — Created by Tessent TestKompress in the IP Creation Phase, this file contains test procedure steps for the Pattern Generation Phase.
 - **Library** — This file contains model descriptions for all library cells used in your design.

Tessent FastScan and Tessent TestKompress Outputs

Tessent FastScan and Tessent TestKompress produce the following outputs:

- **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test pattern formats, refer to the [write_patterns](#) command reference page within this manual, or the “[How to Save the Test Patterns](#)” section in the *Tessent Scan and ATPG User’s Manual*.
- **ATPG Information Files** — These files contain session information that you can save using various commands.
- **Fault List** — This is an ASCII file that contains internal fault information in the standard Mentor Graphics fault format.

Tessent TestKompress produces the following outputs:

- EDT IP Creation Phase:
 - **EDT IP Files** — These files contain the EDT IP described in Verilog or VHDL RTL, as well as connectivity and a black box representation of the core design.
 - **Design Compiler Synthesis Script** — This is a synthesis script, typically used as a template for synthesizing the EDT IP into the design.
 - **EDT Dofile** — This file contains setup commands for the EDT Pattern Generation Phase.
 - **EDT Test Procedure File** — This file contains test procedure steps for the Pattern Generation Phase.
 - **Bypass Files** — These files consist of a dofile and a test procedure file for optionally performing Tessent FastScan ATPG on the design with EDT.

- EDT Pattern Generation Phase:
 - **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test pattern formats, refer to the [write_patterns](#) command description in this manual.

Tessent IJTAG

Tessent IJTAG is Mentor Graphics implementation of the IEEE 1687-2014 (IJTAG) for PDL command retargeting and ICL extraction.

For complete information, refer to [IEEE 1687-2014](#) (IJTAG) term in the *Tessent Glossary* and to the [Tessent IJTAG User's Manual](#).

Tessent IJTAG Inputs

Tessent IJTAG produces the following input:

- **ICL Files** — Instrument Connectivity Language (ICL) descriptions of your design.

Tessent IJTAG Outputs

Tessent IJTAG produces the following outputs:

- **Test Patterns** — Retargeted PDL patterns in many different ATE tester specific formats and Verilog HDL simulation test benches.
- **Extracted ICL Files** — Top-level ICL expressing the interconnection of IJTAG instruments within a design netlist.

Tessent LogicBIST

Tessent LogicBIST contains all the features that are needed to implement Hybrid TK/LBIST Flow in a design.

The product includes the following features:

- Insertion of DFT into the design to make it BIST-ready. This involves scan stitching, x-bounding, and bounding of multicycle and false paths.
- Test point insertion to improve the random pattern testability of the design.
- Hybrid TK/LBIST generation and insertion into the incoming design. This includes sharing the logic between the EDT and LBIST controllers, thereby reducing the overall area overhead. Additionally, the tool also generates logic that allows you to do low power shift during LBIST.
- IJTAG network insertion and extraction, which is necessary to set up Logic BIST and observe the MISR signature.

- Fault simulation of the pseudo-random patterns applied during an LBIST session.
- Generation of top-level patterns that allows merging of patterns for multiple LBIST controllers.

Tessent MemoryBIST

Tessent MemoryBIST provides a complete solution for at-speed testing, diagnosis, repair, debug, and characterization of embedded memories. The solution's architecture is hierarchical, allowing BIST and self-repair capabilities to be added to individual cores as well as at the top level.

For complete information, refer to the [*Tessent MemoryBIST User's Manual*](#).

Tessent MemoryBIST Inputs

To insert memory BIST into your design, you must have either an RTL or a gate-level netlist as well as memory BIST libraries.

If the design contains standard cells, the Tessent cell library or ATPG library is also required. For other IP blocks that do not have either RTL or library models, the simulation model can be loaded using the `read_verilog` or `read_vhdl` command with the `-interface_only` option. This instructs the tool to ignore the internals of all modules specified in the filename argument and extract only the module port definitions and parameters.

Tessent MemoryBIST Outputs

Using Tessent MemoryBIST, you can perform the following:

- Test multiple memories using one memory BIST controller that has one or more BIST steps, where BIST steps are run in sequence.
- Test memories in parallel in one BIST step or in sequence in several BIST steps.
- Define one or more custom test memory algorithms that will be hard coded into the memory BIST controller.
- Choose memory test algorithms from Mentor Graphics' library of algorithms to be hard coded into the memory BIST controller.
- Run the memory BIST controller for all steps with the specific algorithms assigned at generation time (default configuration).
- Run the memory BIST controller in diagnostic mode where you can freeze on a specific BIST step, specific memory test port, or specific error count.
- Select a hard-coded memory BIST algorithm to be applied to a specific memory BIST step at the tester.

- Select an algorithm from a library of algorithms to be applied to a specific memory BIST step.
- Define a custom algorithm at tester time to be applied to a specific memory BIST step.
- Perform repair analysis on memories implementing different redundancy schemes such as row only, column only, or row and column.

Tessent MissionMode

Tessent MissionMode enables communication between any external CPU bus and all on-chip test resources for system-controlled on-line test and diagnosis. The on-chip test resources can consist of any Tessent BIST or DFT capabilities (such as MemoryBIST, LogicBIST or EDT logic) or any third-party IJTAG-compliant instruments.

This product is part of the Mentor Safe tool suite that enables ISO 26262-compliant designs for advanced driver-assistance systems (ADAS) and autonomous driving capabilities for passenger cars, as well as for other safety-critical applications such as aerospace and medicine. The tool targets IC reliability by providing advanced self-test and monitoring capabilities once the IC is packaged and deployed in the device.

Tessent Scan

Tessent Scan tool-specific features include features that support scan insertion.

Specifically, Tessent Scan supports the following functionality:

- Support for full-scan identification and insertion.
- Support for Mux-scan methodology.
- Powerful design rules checking.
- Automatic generation of the scan setup dofile and the test procedure files for use downstream in your flow with Tessent FastScan and Tessent TestKompress ATPG products.
- Display of a variety of information—from design and debugging information to statistical reports for the test set you generate.

Tessent Scan Inputs

Tessent Scan has the following inputs:

- **Design Netlist** — In Verilog format.

- **Test Procedure File** — *Required* if you have existing scan circuitry in your design. The test procedure file defines the operation of existing scan circuitry. See also “[How to Specify Existing Scan Information](#)” in the *Tessent Scan and ATPG User’s Manual*.
- **DFT Library** — Contains the model descriptions for all library cells used in your design, along with the model descriptions for all the scan replacement cells.
- **Command Dofile** — A text file containing a list of Tessent Scan commands. You can also enter these commands interactively.

Tessent Scan Outputs

Tessent Scan produces the following outputs:

- Design Netlist — A scan version of your design netlist that can be in Verilog format.
- **ATPG Setup Files** — The test procedure file, which defines the operation of the scan circuitry in your design, and a dofile for setting up the design and scan circuitry information for ATPG operations.

Tessent ScanPro

Tessent ScanPro includes features that are necessary for scan insertion (part of Tessent Scan), as well as all the advanced DFT insertion features to facilitate scan testing and Logic BIST.

Specifically, Tessent ScanPro supports the following functionality:

- Includes all features that are part of Tessent Scan.
- Generation of test points for pattern reduction as well as test coverage improvement.
- Ability to identify and bound all Xs in the designs. This is needed to reduce the number of Xs that can propagate into the design and can affect compression as well as Logic BIST runs.
- Ability to read in SDC and add DFT to block transitions propagated through Multicycle Paths (MCPs) and False Paths (FPs) from being captured on scan cells.

Tessent ScanPro Inputs

These are the same as Tessent Scan.

Tessent ScanPro Outputs

These are the same as Tessent Scan, with the addition of the following:

- Scan Setup Files — This is the dofile and test procedure file that are required to perform incremental scan stitching of the test point flops.

- Test Point Dofile — This is the file that writes out all the test points that the tool is ready to insert into the design. This is needed when you want just to perform the analysis in the current session but would like to get back to insertion at a later time.

Tessent SiliconInsight

Tessent SiliconInsight provides interactive capabilities for testing, debugging, and characterizing BIST-tested memories and logic, IEEE 1687 IJTAG instruments, and ATPG patterns. It also facilitates on-tester data collection and diagnosis for BIST-tested memories and logic.

For complete information, refer to the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Tessent SiliconInsight operates within the Tessent Shell environment. The tool uses an open ATE interface to interact with embedded instruments for both pre-silicon, desktop, and ATE applications. The open interface is called the Characterization and Debug Package (CDP). The CDP contains test patterns, procedural data and side files that allow access and control of any instrument at any hierarchy level.

For more detailed characterization activities, you can also control instruments such as power supplies and clock generators with interface buses implemented according to the General Purpose Interface Bus (GPIB) standard IEEE-488. To do this, connect a USB-to-GPIB adaptor to a GPIB instrument, which you then connect to the performance board.

Tessent DesignEditor

Tessent DesignEditor features contain a subset of Tessent Shell features.

The product includes the following features:

- Access to design introspection and editing features while consuming a less expensive product license.
- Access to the commands that can operate in the dft context (commands labeled “dft” or “all contexts”).

Tessent DesignEditor Inputs

Tessent DesignEditor has the following inputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.
- **Library** — This file contains model descriptions for all library cells used in your design.

Tessent DesignEditor Outputs

Tessent DesignEditor produces the following outputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.

Chapter 2

Tool Invocations

Tool invocation commands invoke tools that create, edit, manipulate, or diagnose test patterns for DFT applications. These commands are issued from the input window of your Linux workstation and, by default, open in a command line session.

Tool Invocation Command Descriptions	68
create_skeleton_design	69
stil2mgc	71
tessent	75

Tool Invocation Command Descriptions

The notational conventions used to describe the tool invocation commands are the same for all products. Do not enter any of the special notational characters (such as, {}, [], or |) when typing the command.

[Table 2-1](#) describes the notational conventions used in this manual.

Table 2-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-SStatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode Uncollapsed Collapsed	A boldface font indicates a required argument.
[]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
<i>Italic</i>	dofile <i>filename</i>	An italic font indicates a user-supplied argument.
{ }	add_ambiguous_paths { <i>path_name</i> -All} [-Max_paths <i>number</i>]	Braces enclose arguments to show grouping. Do not enter the braces.
	add_ambiguous_paths { <i>path_name</i> -All} [-Max_paths <i>number</i>]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort ON OFF	An underlined item indicates either the default argument or the default value of an argument.
...	add_clocks off_state <i>pin_list</i> ...	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

[Table 2-2](#) contains a summary of the tool invocation commands described in this chapter.

Table 2-2. Tool Invocation Command Summary

Command	Description
create_skeleton_design	Generates files needed to create Tessent TestKompress logic at the RTL stage.
stil2mgc	Invokes stil2mgc to convert dofiles and test procedure files for DFT applications.
tessent	Invokes a Tessent tool in a command-line session.

create_skeleton_design

Invokes Tessent TestKompress and creates files necessary to integrate Tessent TestKompress logic (TestKompress logic) into a design at the RTL stage.

Usage

```
create_skeleton_design -i skeleton_design_input_file
    [-o output_file_prefix]
    [-simulation_library]
    [-design_interface file_name]
    [-module_name module_name]
    [-help [-verbose]]
```

Description

Invokes Tessent TestKompress and creates files necessary to integrate Tessent TestKompress logic (TestKompress logic) into a design at the RTL stage.

This command requires a special design skeleton input file that you create manually. For more information about this file, see “[Integrating Compression at the RTL Stage](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- `-i skeleton_design_input_file`

A required switch and string pair that specifies the pathname of the skeleton design input file.

- `-o output_file_prefix`

An optional switch and string pair that specifies a naming prefix for the output files as follows:

`<output_file_prefix>.v`
`<output_file_prefix>.dofile`
`<output_file_prefix>.testproc`
`<output_file_prefix>.atpglib`

By default, *test* is used for the naming prefix.

- `-simulation_library`

An optional switch that creates a Verilog simulation library from the specified cell library.

- `-design_interface file_name`

An optional switch and string pair that specifies the pathname of the file containing a Verilog description of the design interface. The Verilog description file is used when the TestKompress logic is placed externally to the design core.

- **-module_name *module_name***

An optional switch and string pair that specifies the name of the generated skeleton design module. When this switch is not specified, the default module name is “skeleton_design_top”.

This switch is useful when a design created using the skeleton flow contains multiple EDT blocks. Normally, this would result in all of the EDT blocks having the same name (i.e. skeleton_design_top_edt). You can use this switch during the skeleton design generation step to provide different names for different EDT blocks.

- **-help**

An optional switch that lists all the invocation switches.

- **-verbose**

An optional switch that, together with -help, lists all the invocation switches and a brief description of each.

stil2mgc

Scope: Converts STIL files to dofiles and test procedure files for Tessent FastScan, Tessent TestKompress, and Tessent Scan.

Invokes stil2mgc and produces a dofile for Tessent FastScan, Tessent TestKompress, and Tessent Scan that defines clocks, scan chains, scan groups, and pin constraints.

Usage

```
stil2mgc {-STil stil_filename [-TPf tpf_filename] [-DOfile dofile_name]
          [-FLex_dofile dofile_name]
          [-Alias Min | All] [-CAapture NOne | Single | NAmed]
          [-EDGE_strobe_processing ON | OFF]
          [-INFer_pulse_clock ON | OFF]
          [-EXec exec_name]
          [-WRITE_CORE_DESCRIPTION
              [-OUTPUT_DIRECTORY dir_path]
              [-TCD_SCAN_EXTENSION ext]
              [-CTL_mode mode_name]
              [-CORE_name name]
          ]
          [-LOGfile logfile_name [-REplace]]} |
          [-help | -Usage | -MANual | -Version]
          [-Verbose]
```

Description

Invokes stil2mgc and produces a dofile for Tessent FastScan, Tessent TestKompress, and Tessent Scan that defines clocks, scan chains, scan groups, and pin constraints. This tool also creates test procedure files with a timeplate and the following standard scan procedures: test_setup, load_unload, and shift.

This command translates STIL signal groups, which are used in timeplates or procedures, into “alias” statements in the test procedure file. It also produces a test procedure file timeplate definition for each WaveformTable in the STIL file, exactly matching the timing specified in the STIL file.

Any procedure or macro with a name that matches a Mentor Graphics procedure type (load_unload, master_observe) is translated into that Mentor Graphics procedure. Any procedure or macro in the STIL file that has “capture” in its name is translated into a named capture procedure when you use the “-Capture Named” switch.

Any other macro whose name does not match anything that Mentor Graphics uses is translated into an unused sub-procedure.

For more information about translation details, refer to “[Notes About Using the stil2mgc Tool](#)” in the *Tessent Shell User’s Manual*.

Arguments

- **-STil *stil_filename***
A required switch and string pair that specifies the name of the input STIL file. The supplied STIL file may be a STIL Procedure File (SPF) or a Core Test Language (CTL) file.
- **-TPf *tpf_filename***
An optional switch and string pair that specifies the name of the test procedure file to generate. If not specified, the *tpf_filename* will be named by appending “.proc” to the STIL Test Procedure file name.
- **-DOfile *dofile_name***
An optional switch and string pair that specifies the name of the dofile to generate. If not specified, the *dofile_name* will be named by appending “.dof” to the STIL Test Procedure file name.
- **-FLex_dofile *dofile_name***
An optional switch and string pair that produces an additional dofile targeted at the FlexTest tool.
- **-ALias Min | All**
An optional switch and literal pair that produces the minimum required alias statements in the MGC Procedure file or to produce alias statements for all signal groups found in the STIL Test Procedure file. The default is to produce the minimum required alias statements.
- **-CAapture NOne | Single | NAmed**
An optional switch and literal pair that adds information to the MGC Procedure file. This switch will either produce no capture procedures, a single default capture procedure, or multiple named capture procedures for each capture Macro found in the STIL Test Procedure File. The default is to produce no capture procedures.
- **-EDGe_strobe_processing OFF | ON**
An optional switch and literal pair that specifies the type of strobe processing, edge strobe or window strobe, used to measure events.

ON

When you specify ON, the tool will process H and L waveform events in the SPF file to indicate edge strobe timing.

OFF

When you specify OFF, edge strobe timing is not created. This is the default

- **-INFer_pulse_clock ON | OFF**

An optional switch and literal pair to specify that stil2mgc uses the generic pulse_clock statement for clocks whose timing edges are identical.

ON

When you specify ON and the timing edges for all of the clock edges in the SPF file are identical, the produced timeplates uses the pulse_clock statement. If the timing edges are not identical for each clock, the produced timeplate uses individual pulse statements. This is the default

OFF

When you specify OFF, individual pulse clock statements are used for each clock.

- **-EXec *exec_name***

An optional switch and string pair to specify the name of the STIL PatternExec used when the STIL file has more than one PatternExec block. An error is reported when this switch is missing and more than one PatternExec blocks exist in the STIL file.

- **-WRITE_CORE_DESCription [-OUTPUT_DIRectory *dir_path*]
[-TCD_SCAN_EXTension *ext*] [-CTL_mode *mode_name*]
[-CORE_name *name*]**

An optional switch and options that write sub-chain definitions into the *tcd_scan* file. If you specify this switch but do not provide the CTL block in the input file, then the tool issues an error. Choose any of the following options:

-OUTPUT_DIRectory *dir_path* — An optional switch and string pair that specifies the destination directory where the tool writes the TCD files. By default, the tool writes the generated TCD files to the current directory. If the specified directory path does not exist, then the tool creates the directory.

-TCD_SCAN_EXTension *ext* — An optional switch and string pair that specifies the file extension of the generated scan TCD file. The default file extension is “*tcd_scan*”. The tool creates the output file name by appending the extension to a core name.

-CTL_mode *mode_name* — An optional switch and string pair that specifies the test mode name corresponding to the scan chain information that the tool reads.

-CORE_name *name* — An optional switch and string pair that specifies a core name from available values.

- **-LogFile *logfile_name***

An optional switch and string pair that specifies the name of the file to write all session information.

- **-REplace**

An optional switch that overwrites the -LogFile *logfile_name* if one by the same name already exists.

- **-help**

An optional switch that displays a message that contains all the stil2mgc switches and a brief description of each.

- **-Usage**

An optional switch that displays the usage syntax for the stil2mgc command.

- **-MANual**
An optional switch that opens the Tessent Documentation System.
- **-Version**
An optional switch that displays the version of the stil2mgc tool that you currently have available.

Examples

Example 1

The following example executes the stil2mgc tool on *design.spf*.

```
shell> stil2mgc -stil design.spf -tpf new_design.tpf
      -dofile new_dofile.do
```

Example 2

The following example demonstrates invoking the tool for single-mode output:

```
stil2mgc -stil ../data/multi_mode.ctl \
      -ctl_mode scan_mode \
      -write_core_description
```

In the example, the tool writes the tool-generated files to the current directory.

Example 3

This example demonstrates multi-mode output invocation. In the example, the tool writes the generated file as *./outDir/block.tcd*. This file contains description of all modes containing valid scan chain data.

```
stil2mgc -stil ../data/multi_mode.ctl \
      -output_directory ./outDir \
      -tcd_scan_extension tcd \
      -core_name block \
      -write_core_description
```

Example 4

The following example sources Tessent Shell tool-generated files:

```
set_context dft -scan
# (...) loading design
set_design_sources -format tcd_scan -Y ./ -extensions tcd_scan
```

tessent

Invokes a Tessent tool in a command-line session.

Usage

```
tessent {-Shell
          | {-DIAGSERVER diagnosis_arguments}
          | {-YIELDINSIGHT [-NOGui]}
          | {-Siliconinsight {-Desktop | -Ate}}
          | -DOFile dofile_name... [-HISTORY]
          | [-Arguments name=value ...]
          | [-LOGfile logfile_name [-REPlace]]
          | [-LICENSE_wait {minutes | NO_Queue | UNLimited}]
          | [-underscore_commands_only]
          | [-ignore_startup_file]
          | [-Help | -Usage | -Manual | -Version]}
```

Description

Invokes a Tessent tool in a command-line session.

Arguments

- **-Shell**

A switch for invoking Tessent Shell. For more information, refer to the “[Tessent Shell Introduction](#)” in the *Tessent Shell User’s Manual*.

- **-DIAGSERVER**

A switch for invoking Tessent Diagnosis server. See “[Running Tessent Diagnosis Server](#)” in the *Tessent Diagnosis User’s Manual*.

- **-YIELDINSIGHT [-NOGUI]**

A switch for invoking Tessent YieldInsight. See the “[Overview of Tessent YieldInsight](#)” in the *Tessent YieldInsight User’s Manual* for complete information.

An optional -NOGUI switch invokes the tool in command-line mode. By default, Tessent YieldInsight invokes in graphical user interface (GUI) mode.

- **-Siliconinsight {-Desktop | -Ate}**

A switch and literal pair for invoking Tessent SiliconInsight. Choose from one of the following options below:

- **-Desktop** — Invoke Tessent SiliconInsight Desktop.
- **-Ate** — Invoke Tessent SiliconInsight ATE.

- **-DOFile *dofile_name...* [-HISTORY]**

An optional switch and repeatable string pair that specifies the name of one or more dofiles to execute upon invocation. As the tool is processing each dofile, the Tcl variable

“`tessent_user_argv0`” is assigned the name of the current dofile. `-HIStory` is an optional switch that adds dofile commands to the command line history list. The tool ignores the `-History` switch if you issue it without the `-Dofile` switch.

- `-Arguments name=value ...`

An optional switch and repeatable name/value pair that specifies variable names and values in a list named “`tessent_user_arg`.” No spaces are allowed between the name, the equal sign, and the value. This list can later be accessed by a dofile for the purpose of controlling parameters in the dofile. For more information, refer to “[Example 2](#)”.

- `-LOGfile logfile_name [-REPlace]`

An optional switch and string pair that specifies the name of the file to which you want Tessent Diagnosis to write all session information. The default is to display session information to the standard output. The version banner that indicates the tool, version, date, and platform on which the log file was produced is included at the beginning of the file.

An optional `-REPlace` switch overwrites the `logfile_name` if a log file of the same name already exists.

- `-LICense_wait minutes | NO_Queue | UNLimited`

An optional switch and integer, or switch and literal, that specifies the tool’s response if the license is unavailable.

Choose one of the following options:

- `minutes` — A positive integer that specifies the number of minutes to wait for the license.
- `NO_Queue` — A literal that directs the Tessent tool to exit immediately if no license is available.
- `UNLimited` — A literal that directs the Tessent tool to wait with no time limit for a license. This is the default.

- `-underscore_commands_only`

An optional switch that directs the tool to only allow commands that use the underscore style. This option disables the legacy commands that used spaces. For example, if this option is specified, the tool would accept `analyze_drcViolation` but would not accept “`analyze drc violation`”.

- `-ignore_startup_file`

An optional switch that instructs the tool to not read the `.tessent_startup` file during invocation.

- `-Help | -Usage | -Manual | -Version`

Optional switch choices that report the following:

- `-Help` — An optional switch that lists the switches for the tessent command.
- `-Usage` — An optional switch that lists the switches for the tessent command.

- -Manual — An optional switch that opens the Tessent Documentation System.
- -Version — An optional switch that displays the version of the tessent tool you specify.

Examples

Example 1

The following example invokes Tessent Shell, uses the logfile *myLogFile* replacing any existing logfile with the same name, and executes the dofile *myDofile*.

```
tessent -shell -logfile myLogFile -replace -dofile myDofile
```

Example 2

The following example invokes Tessent Shell and runs a dofile named *dofile2*. The command line also specifies three variable/value pairs that are placed in a Tcl list named “*tessent_value_arg*” and thus made available to the dofile. It shows how to use single and double quotes to set variables that contains spaces. The system shell will not perform the expansion environment variable of the text surrounded by single quotes thus preserving the \$, unlike the text in double quotes:

```
% setenv I2 3% tessent -shell -dofile dofile2 -arguments depths="9 \
$I2" comment='Cost $US' output_dir=myOutDir
```

The following is the listing of *dofile2*, which demonstrates how to use the -arguments switch to set the values of variables inside the dofile. This technique allows you to reuse dofiles without modification:

```
# dofile2
# Set the default values of the control parameters
array set params {
    depths {}
    comment ""
    output_dir "outDir"
}
array set params $tessent_user_arg
# Use some params
puts "comment = $params(comment)"
if { [llength $params(depths)] > 0} {
    # do whatever needs to be done in this case
    puts "depth list = $params(depths) "
}
if { ![file isdirectory $params(output_dir)] } {
    puts "Creation of directory $params(output_dir)"
    file mkdir $params(output_dir)
}
```

Here is the transcript produced by the invocation of Tessent Shell.

```
% setenv I2 3% tessent -shell -dofile dofile2 -arguments depths="9 $I2" \
comment='Cost $US' output_dir=myOutDir
```

```
// Copyright 2011-2014 Mentor Graphics Corporation
// ... truncated ...
//
// command: # dofile2
// command: # Set the default values of the control parameters
// command: array set params {
//           depths {}
//           comment ""
//           output_dir "outDir"
//         }
// command: array set params $tessent_user_arg
// command: # Use some params
// command: puts "comment = $params(comment)"
//           comment = Cost $US
// command: if { [llength $params(depths)] > 0 } {
//           # do whatever needs to be done in this case
//           puts "depth list = $params(depths)"
//         }
//         depth list = 9 3
// command: if { ![file isdirectory $params(output_dir)] } {
//           puts "Creation of directory $params(output_dir)"
//           file mkdir $params(output_dir)
//         }
//         Creation of directory myOutDir
```

Example 3

The following example shows how to create a single shell script that invokes and passes arguments to Tessent Shell, which then executes the commands in the shell script as a regular dofile. In order for the following to work, the shell script file must have executable permissions.

The contents of the shell script, called *tScript* in this example, are as follows.

```
#!/bin/sh
#\ 
exec tessent -shell -dofile "$0" -arguments ${1+"$@"}

set_transcript_style off
puts "$tessent_user_arg"
# add dofile commands here
exit
```

The Linux shell interprets the second line ('#\') as a comment and then executes the third line, which invokes Tessent Shell and passes the entire shell script as a dofile. (Tessent Shell tool's Tcl interpreter ignores the second and third lines of the shell script and then interprets all following lines as normal Tessent Shell commands.)

The -arguments switch on the third line allows you to pass an arbitrary number of argument/value pairs from the shell script invocation into a standard Tcl list called “*tessent_user_arg*” for later parsing and use by Tessent Shell. So, for example, when you invoke the shell script as follows:

```
% tScript myarg1=myvalue1 myarg2= myarg3=myvalue3
```

The three arguments and their values are passed to Tessent Shell, and the relevant part of the transcript would appear as follows:

```
// command: #!/bin/sh
// command: # exec tessent -shell -dofile "$0" -arg ${1+"$@"}
// command: set_transcript_style off
myarg1 myvalue1 myarg2 {} myarg3 myvalue3
```


Chapter 3

Command Dictionary (A - D)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual.

Table 3-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode <u>Uncollapsed</u> Collapsed	A boldface font indicates a required argument.
[]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
<i>Italic</i>	dofile <i>filename</i>	An italic font indicates a user-supplied argument.
{ }	add_ambiguous_paths { <u>path_name</u> -All} [-Max_paths number]	Braces enclose arguments to show grouping. Do not enter the braces.
	add_ambiguous_paths { <u>path_name</u> -All} [-Max_paths number]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort <u>ON</u> OFF	An underlined item indicates either the default argument or the default value of an argument.
...	add_clocks off_state <i>pin_list</i> ...	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Command Descriptions

This section describes commands, alphabetically from A through D, that available in Tessent Shell. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 3-2. Commands A Through D

Command	Description
add_ambiguous_paths	Derives multiple paths from ambiguous paths when there is path or edge ambiguity.
add_atpg_constraints	Restricts all patterns placed into the internal pattern set according to the user-defined constraints.
add_atpg_functions	Creates an ATPG function that you can then use when generating user-defined ATPG constraints.
add_bist_capture_range	Associates a named ATPG capture procedure with a subset of the BIST patterns.
add_black_boxes	Defines instances of Verilog modules or cell library models as black boxes and sets the constrained value on output or bidirectional black box pins.
add_browser_data	Adds the specified data to the active tab of the DFTVisualizer Browser window.
add_capture_handling	Specifies the data capturing behavior for the given state element.
add_cdp_test	Creates a new test in the CDP.
add_cell_constraints	Constrains scan cells (also non-scan cells for CX, TX, DX, and SX constraints) to a constant value.
add_cell_models	Specifies DFT library models for user-defined test points, system-generated test points, and system-generated test logic.
add_chain_masks	Masks the load, capture, and/or unload values on the specified scan chains during simulation.

Table 3-2. Commands A Through D (cont.)

Command	Description
<code>add_clocks</code>	Adds scan or non-scan clocks to the clock list for proper scan operation.
<code>add_config_element</code>	Adds a configuration element in the configuration data.
<code>add_config_message</code>	Adds error, warning, or information messages to configuration elements.
<code>add_config_tab</code>	Adds one configuration tree tab to the DFTVisualizer Configuration Data window.
<code>add_control_points</code>	Specifies user-defined control points during test point insertion.
<code>add_core_instances</code>	Adds a core instance to the design by associating/binding the core description currently in memory with the specified core instance(s) in the design.
<code>add_dft_clock_enables</code>	Defines a pin or port of the current design or a port of a submodule as a clock enable signal that, when activated, disables the clock gating on a clock branch.
<code>add_dft_clock_mux</code>	Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.
<code>add_dft_control_points</code>	Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the process_dft_specification command.
<code>add_dft_modal_connections</code>	A command used to insert modal connections from specified output data source nodes or to input data destination nodes typically used to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports.
<code>add_dft_signals</code>	Adds static and dynamic DFT signals used to control various aspect of the DFT logic.
<code>add_display_callout</code>	Displays a text string in a callout box on a specified object in the schematic window.
<code>add_display_data</code>	Adds specified data columns to the Data window of DFTVisualizer, opening the Data window if it is not already open. If data corresponding to an added column is available, it is displayed.

Table 3-2. Commands A Through D (cont.)

Command	Description
add_display_instances	Displays instances in DFTVisualizer and enables you to trace visually through the design using the Flat Schematic or Hierarchical Schematic window.
add_display_path	Displays specified objects or paths in the schematic window.
add_drcViolation	Adds a DRC violation to the list of DRC violations for a given DRC rule.
add_edt_blocks	Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.
add_edt_configurations	Sets up EDT logic with two compression configurations.
add_false_paths	Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.
add_fault_sites	Automatically adds fault site definitions for library cells that do not have UDFM definitions.
add_faults	Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected (actual category is UC).
add_icl_ports	Specifies top design ports that will be added in the ICL file generated during ICL extraction.
add_icl_scan_interfaces	Adds the specified scan interfaces to the new ICL top module that will be created during ICL extraction.
add_iddq_exceptions	Enables you to specify individual sites where the tool does not apply the Iddq restrictions.
add_ijtag_logical_connection	Creates a new logical connection path through the current design by specifying the source and destination of the logical path.
add_input_constraints	Constrains primary inputs to specified values.
add_layout_core_information	In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB.
add_lfsr_connections	Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).
add_lfsr_taps	Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).
add_lfsrs	Adds LFSRs for use as PRPGs or MISRs.

Table 3-2. Commands A Through D (cont.)

Command	Description
add_lists	Adds pins to the list of pins on which to report.
add_loadboard_loopback_pairs	Creates a loopback connection from specified output ports to specified input ports.
add_misrs	Specifies the MISR structure and initial seed.
add_nofaults	Places nofault settings either on pin pathnames, pin names of specified instances, or modules.
add_noscan_instances	Sets a user-specified is_non_scannable attribute on the specified design objects.
add_notest_points	Prevents test logic insertion in the specified regions.
add_observe_points	Specifies user-defined observe points during test point insertion.
add_output_masks	Ignores any fault effects that propagate to the primary output pins you name.
add_primary_inputs	Adds a primary input (PI) also known as an input pseudo_port to the specified pins or nets.
add_primary_outputs	Adds primary outputs to the specified net.
add_processors	Enables the tool to run multiple processors in parallel on multiple machines to reduce runtime.
add_read_controls	Specifies the off-state value for specified RAM read control lines.
add_register_value	Creates a static or dynamic variable by allowing you to specify a register identifier, value_name, and, optionally, the state string, value_string.
add_rtl_to_gates_mapping	Adds name-mapping rules for RTL to post-synthesis/post-layout name mapping.
add_scan_chains	Specifies a name for a pre-existing scan chain within the design.
add_scan_groups	Specifies a group name for a set of pre-existing scan chains in a design.
add_scan_instances	Adds specified instances to the scannable instance list.
add_scan_mode	Specifies a chain scan mode, which defines one or more scan chains.
add_scan_segments	Adds one or more existing scan segment(s).
add_simdut_fault	Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.

Table 3-2. Commands A Through D (cont.)

Command	Description
<code>add_simulation_context</code>	Creates a new user-defined simulation context.
<code>add_simulation_forces</code>	Forces one or more gate_pin objects to the specified value.
<code>add_synchronous_clock_group</code>	Defines a set of clocks that are synchronous (compatible) with each other.
<code>add_tied_signals</code>	Adds a value to floating signals or pins.
<code>add_to_collection</code>	Adds objects to a base collection and returns the new collection without affecting the original base collection.
<code>add_write_controls</code>	Specifies the off-state value for specified RAM write control lines.
<code>analyze_atpg_constraints</code>	Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.
<code>analyze_bus</code>	Analyzes the specified bus gates for contention problems.
<code>analyze_compression</code>	Analyzes the compression for an application in two steps.
<code>analyze_control_signals</code>	Identifies and optionally defines the primary inputs of control signals.
<code>analyze_drcViolation</code>	Generates a schematic in the schematic windows the portion of the design involved with the specified rule violation number.
<code>analyze_fault</code>	Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in DFTVisualizer.
<code>analyze_graybox</code>	Identifies the instances and nets to be included in the graybox netlist by automatically setting their <code>in_graybox</code> attributes to “true”. Subsequently, a “ <code>write_design-graybox</code> ” command writes out the graybox netlist for the marked instances and nets.
<code>analyze_restrictions</code>	Performs an analysis to automatically determine sources of the problem from a failed ATPG run.
<code>analyze_scan_chains</code>	Distributes scan elements into new scan chains.
<code>analyze_simulation_mismatches</code>	Analyzes simulation mismatches that occur between test patterns and a simulation performed by a specified external timing simulator.
<code>analyze_test_points</code>	Specifies the test points analysis and generates a list of test points to be inserted by the tool.
<code>analyze_wrapper_cells</code>	Identifies shared and dedicated wrapper cells for the primary I/O ports.

Table 3-2. Commands A Through D (cont.)

Command	Description
analyze_xbounding	Performs X-bounding analysis.
annotate_diagnosis	Adds DFM and RCD data, if available, to existing diagnosis reports.
append_to_collection	Adds one or more objects to a collection referenced by the specified var_name.
apply_specification_defaults	Automatically applies all defaults specified in the DefaultsSpecification/DftSpecification and/or DefaultsSpecification/PatternsSpecification wrappers to wrappers that were created manually.
catch_output	Executes the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.
check_design_rules	Transitions the tool from setup mode to analysis mode.
check_synthesis	Checks the status of synthesis that was previously launched by the run_synthesis command.
check_testbench_simulations	Checks the status of simulations that were previously launched by the run_testbench_simulations command.
close_layout	Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.
close_pattern_set	Finalizes and closes the currently open pattern_set.
close_tsdb	Makes the contents of a previously opened TSDB directory invisible to the tool.
close_visualizer	Saves display data and closes DFTVisualizer.
compare_collections	Compares the contents of two collections.
compress_layout	Compresses a LDB.
compress_patterns	Compresses patterns in the current test pattern set.
copy_collection	Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.
copy_module	Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.
copy_simulation_context	Copies the simulation values and forces from one simulation context to another.
create_bisr_segment_order_file	Creates the BisrSegmentOrderSpecification configuration data wrapper and extracts the list and ordering of BISR chain segments.

Table 3-2. Commands A Through D (cont.)

Command	Description
create_capture_procedures	Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.
create_connections	Connects pin, net, or port objects.
create_dft_specification	Creates the “DftSpecification(<i>design_name,id</i>)” configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.
create_diagnosis_patterns	Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.
create_feature_statistics	Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.
create_flat_model	Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the DFTVisualizer Flat Schematic window.
create_icl_verification_patterns	Specifies the creation of a structural pattern to verify the proper operation of the IJTAG network.
create_initialization_patterns	Creates RAM initialization patterns and places them in the internal pattern set.
create_instance	Instantiates an instance of a module mod_spec inside a design module in the current design.
create_layout	Generates a diagnosis tool-compatible Layout Database directory (LDB) from LEF/DEF input files.
create_module	Creates a new design module with no ports.
create_net	Creates a net inside an instance of a design module in the current design.
create_patterns	Performs automatic high-speed test generation and pattern compression.
create_patterns_specification	Generates a pattern specification for the specified usage. The usage is either signoff or manufacturing.
create_pin	Creates a pin on an instance of a design module.
create_port	Creates a port on the specified design module.
create_scan_chain_family	Creates a scan_chain_family object, which controls how one or more scan chains will be allocated/connected during distribution.

Table 3-2. Commands A Through D (cont.)

Command	Description
create_silicon_insight_setup_specification	Generates a default Tessent SiliconInsight Setup Specification.
delete_atpg_constraints	Removes state restrictions from the specified pins. You define pin restrictions by using the add_atpg_constraints command.
delete_atpg_functions	Removes the specified function definitions.
delete_bist_capture_ranges	Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.
delete_black_boxes	Undoes the effect of the add_black_boxes command.
delete_browser_data	Removes data from the active tab of the DFTVisualizer Browser window.
delete_capture_handling	Removes the special data capture handling for the specified objects.
delete_capture_procedures	Removes the specified capture procedures from internal memory.
delete_cdp_test	Removes the specified test from the CDP.
delete_cell_constraints	Removes constraints placed on scan cells.
delete_cell_library	Removes all current cell libraries, the current design including the flat model, and all tool data that is design-specific.
delete_cell_models	Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.
delete_chain_masks	Removes chain masks from the specified scan chains.
delete_clocks	Removes clocks from the clock list.
delete_config_element	Deletes one or more configuration elements. Only elements which can be added can be deleted.
delete_config_messages	Deletes error, warning or info message that were added to configuration elements using the add_config_message command.
delete_config_tabs	Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using the add_config_tab or display_specification command.
delete_connections	Disconnects net, pin, or port objects.

Table 3-2. Commands A Through D (cont.)

Command	Description
delete_core_descriptions	Removes core descriptions that have been read in using the read_core_descriptions command from memory.
delete_core_instances	Removes an instance binding created with the add_core_instances command.
delete_design	Removes all design modules including the flat model and all tool data that is design-specific.
delete_dfm	Deletes a DFM rule from the open LDB.
delete_dft_clock_enables	Deletes one or all previously added DFT clock enable signals.
delete_dft_clock_muxes	Deletes one or all previously added DFT clock multiplexers.
delete_dft_control_points	Deletes one or all previously added DFT control points.
delete_dft_modal_connections	You use the delete_dft_modal_connections command to delete modal connection specifications previously added by the add_dft_modal_connections command.
delete_dft_signals	Deletes a DFT signal that was previously added using the add_dft_signals command.
delete_display_callout	Deletes a callout box from the schematic window.
delete_display_data	Removes a data column from the DFTVisualizer Data window.
delete_display_instances	Removes the specified design instances from a DFTVisualizer display window.
delete_display_path	Deletes specified objects or paths in the schematic window.
delete_edt_blocks	Removes the data for the specified EDT block(s) from the tool's internal database.
delete_edt_configurations	Deletes specified compression configurations from all blocks in a design.
delete_false_paths	Removes definitions of false paths.
delete_fault_sites	Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.
delete_faults	Removes faults from the current fault list.
delete_flat_model	Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.
delete_icl_modules	Deletes the specified ICL modules from memory.

Table 3-2. Commands A Through D (cont.)

Command	Description
delete_icl_ports	Undoes the effect of the add_icl_ports command on a specified list of top level ports.
delete_icl_scan_interfaces	Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that will be created during ICL extraction.
delete_iddq_exceptions	Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.
delete_ijtag_logical_connection	Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.
delete_input_constraints	Removes previously applied constraints from primary input pins.
delete_instances	Deletes the specified instance objects from the current design.
delete_iprocs	Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.
delete_layout_core_information	In hierarchical layout-aware diagnosis, deletes all core instance data or the core instance data from a specified design.
delete_layout_verification	Reports or deletes stored layout verification information from the current layout database.
delete_lfsr_connections	Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).
delete_lfsr_taps	Removes the tap positions from a Linear Feedback Shift Register (LFSR).
delete_lfsrs	Removes the specified Linear Feedback Shift Registers (LFSRs).
delete_lists	Removes pins from the pin list the tool monitors and reports on during simulation.
delete_loadboard_loopback_pairs	Deletes any loopback connection associated with the specified ports.
delete_misrs	Deletes MISRs you specify with the add_misrs command.
delete_multicycle_paths	Removes definitions of multicycle paths previously read in using the read_sdc command.
delete_nets	Removes net objects inside an instance of a design module.

Table 3-2. Commands A Through D (cont.)

Command	Description
delete_nofaults	Removes the nofault settings from either the specified pin or instance/module pathnames.
delete_nonscan_instances	Resets any user-specified is_non_scannable attribute on the specified objects.
delete_notest_points	Allows test logic insertion in the specified regions.
delete_output_masks	Removes the masking of the specified primary output pins.
delete_patterns	Deletes the current pattern set.
delete_pins	Removes pin objects from a design module instance.
delete_ports	Removes port objects from a design module.
delete_primary_inputs	Removes the specified primary inputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.
delete_primary_outputs	Removes the specified primary outputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.
delete_processors	Removes processor definitions previously specified with the add_processors command.
delete_read_controls	Removes the read control line definitions from the specified primary input pins.
delete_register_value	Deletes all or a specified register value variable specified with the add_register_value command.
delete rtl_to_gates_mapping	Removes name-mapping rules for RTL to post-synthesis/post-layout name mapping.
delete_scan_chain_families	Removes scan chain families.
delete_scan_chains	Removes the specified scan chain definitions from the scan chain list.
delete_scan_elements	Removes scan elements (such as chain or segment) of a given name.
delete_scan_groups	Removes the specified scan chain group definitions from the scan chain group list.
delete_scan_instances	Removes the specified, sequential instances from the user-identified scan instance list.
delete_scan_modes	Removes one or more scan modes.

Table 3-2. Commands A Through D (cont.)

Command	Description
<code>delete_simdut_fault</code>	Removes the injected stuck-at fault at the specified signal or all the injected faults.
<code>delete_simulation_contexts</code>	Deletes one or more user-defined simulation contexts.
<code>delete_simulation_forces</code>	Removes forces from one or more gate_pin objects.
<code>delete_synchronous_clock_group</code>	Removes one or more user-defined synchronous clock groups.
<code>delete_test_points</code>	Removes the specified test point definitions.
<code>delete_tied_signals</code>	Removes the assigned (tied) value from the specified floating nets or pins.
<code>delete_write_controls</code>	Removes the write control line definitions from the specified primary input pins.
<code>diagnose_failures</code>	Diagnoses the failures in the specified failure file.
<code>display_diagnosis_report</code>	Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the DFTVisualizer Flat Schematic window.
<code>display_message</code>	Sends a message to the specified message stream.
<code>display_specification</code>	Displays the DftSpecification(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.
<code>dofile</code>	Executes the commands contained within the specified file.

add_ambiguous_paths

Context: dft -scan, dft -edt, dft -test_points, patterns -scan

Mode: analysis

Derives multiple paths from ambiguous paths when there is path or edge ambiguity.

Usage

`add_ambiguous_paths [path_name | -All] [-Max_paths number]`

Description

Derives multiple paths from ambiguous paths when there is path or edge ambiguity.

When paths have path and/or edge ambiguity, by default Tessent FastScan and Tessent TestKompress select a single path that satisfies the pin connectivity within the path definition file. If you want the tool to have additional choices for path and/or edge ambiguity, you can use the `add_ambiguous_paths` command. When Tessent FastScan or Tessent TestKompress select from the ambiguous paths, they only consider the possible connectivity between points and do not attempt to determine whether the edges are sensitive. Moreover, the tool marks all derived paths as unambiguous paths.

Arguments

- *path_name*

A string that specifies the name of an ambiguous path from which you want the tool to derive unambiguous paths and add them to the path list.

- `-All`

A switch that adds up to `-Max_paths` unambiguous paths for each ambiguous path encountered in the path list.

- `-Max_paths number`

An optional switch and integer pair that specifies the maximum number of unambiguous paths you want the tool to derive. If you issue the command without this switch, the default maximum number of unambiguous paths is 10.

Note

 Mentor Graphics recommends you use this switch with a large enough number to ensure you get all the unambiguous paths derivable from any ambiguous path. You will know the specified number is large enough if a message similar to “// All unambiguous paths...were added to the path list” is displayed in the transcript. If the message begins with a number (for example, “// 10 paths... were added to the path list”), it indicates the tool did not get all the derivable unambiguous paths.

Examples

The following example loads in a path definition file and changes the maximum number of paths to five:

```
read_fault_sites /user/design/pathfile  
add_ambiguous_paths -all -max_paths 5
```

Related Topics

[delete_fault_sites](#)
[read_fault_sites](#)
[write_patterns](#)

add_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Restricts all patterns placed into the internal pattern set according to the user-defined constraints.

Usage

```
add_atpg_constraints {0 | 1 | Z} {pin.pathname | gate_id# | function_name |  
{-Cell cell_name {pin_name...}}... [-Instance {object_expression...}]  
[-MODULE {module_name ...}]... [-DYnamic | -SStatic]  
[-NOCapclock_check | -Load_only]
```

Description

Restricts all patterns placed into the internal pattern set according to the user-defined constraints.

When a simulated pattern is rejected, a message is issued that indicates the number of rejected patterns and the first gate at which the failure occurred. You can control the severity of the violation with the [set_contention_check](#) command. If you set the checking severity to Error, the simulation is terminated if it rejects a pattern due to a user-defined constraint. You can analyze the simulation data up to the termination point by using the [report_gates](#) command with the Error_pattern option.

User-defined constraints are used when using Tesson FastScan or Tesson TestKompress to generate test patterns with deterministic test generation methods. When test patterns are generated randomly using Tesson FastScan or Tesson TestKompress, the use of the user-defined ATPG constraints is not ensured; however, non-conforming random patterns are rejected.

If you change an ATPG constraint for a single internal set of patterns, pattern compression continues using the new constraints, which can cause the good patterns to be rejected; therefore, you should remove all ATPG constraints before compressing the pattern set.

Note

 If you constrain a pin by directly creating an ATPG constraint to the *pin.pathname*, and then create another constraint that indirectly creates a different constraint, the constraint that directly specified the *pin.pathname* (overriding the global ATPG cell constraint) is used.

The -Dynamic switch lets you change the ATPG constraints any time during the ATPG process, affecting only the fault simulation and capture that occurs after the constraint changes. Any subsequently simulated patterns that fail to meet the current constraints are rejected during fault simulation.

Dynamic ATPG constraints do not affect DRC because of their temporary nature. Static ATPG constraints are unchangeable in analysis mode, ensuring that DRC must be repeated if they are changed.

Arguments

- **0 | 1 | Z**

A literal that restricts the named object to a low state, high state, or high impedance state, respectively. You must choose one of the three literals to indicate the state value to which you want the tool to constrain the specified object.

The following lists the four objects on which you can place the constraint. You can use any number of the four argument choices in any order.

- ***pin.pathname***

A repeatable string specifying the pathname of a top-level pin or a library model pin on which you are placing the constraint. Pathnames of pins on intermediate hierarchy modules are not supported.

- ***gate_id#***

A repeatable integer specifying the gate identification number of the gate you want to constrain.

- ***function_name***

A repeatable string specifying the name of a function you created with the [add_atpg_functions](#) command. If you place a constraint on an ATPG function that you generated with the add_atpg_functions -Cell command, all cells affected by that ATPG function are also constrained. You can delete all these constraints using the *function_name* argument with the [delete_atpg_constraints](#) command.

- **-Cell *cell_name {pin_name}***

A repeatable switch and string pair specifying the name of a DFT library cell and name of a pin on that cell. You can repeat the *pin_name* argument if there are multiple pins or nets on a cell that you need to constrain. If you use the -Cell option, an ATPG constraint is placed on every occurrence of that cell within the design.

- **-Instance *object_expression***

An optional switch and repeatable string pair that places ATPG constraints inside the specified list of instances. You can use regular expressions, which may include any number of embedded asterisk (*) and/or question mark (?) wildcard characters. You can only use this switch when using the -Cell switch or when a function name is specified.

- **-MODULE *module_name***

An optional switch and repeatable string pair that places the ATPG constraints inside all instances of the specified module.

- **-DYnamic**

An optional switch that satisfies only the ATPG constraints during the ATPG process and not during DRC. You can change these constraints during the ATPG process; therefore, DRC does not check these constraints. This is the default.

- **-SStatic**

An optional switch that satisfies the ATPG constraints you are defining during all its processes. You can only add or delete static ATPG constraints when you are in Setup mode, ensuring the static ATPG constraints are used for all ATPG analyses during design rules checking. DRC checks for any violations of ATPG constraints during the simulation of the test procedures (rule E12).

- **-NOCapclock_check**

An optional switch that suppresses checking of specified ATPG constraints after the capture clock. By default, ATPG constraints are checked at the same time as bus contention. If contention checking is performed after the capture clock, ATPG constraints are also checked after the capture clock. You cannot use this switch and **-Load_only** at the same time.

In some situations, you may have some ATPG constraints that do not need to be checked after the capture clock; although, you may want other constraints and bus contention to be checked after the capture clock. In this case, you can use the **-Nocapclock_check** switch on certain constraints to suppress checking of those constraints after the capture clock.

- **-Load_only**

An optional switch specifying the Atpg constraint only applies to the first cycle after shift. When using Tessent FastScan or Tessent TestKompress, the values loaded into the scan chain and the forced PI values satisfy the constraint; however, once the leading edge of the first clock occurs, the constraint no longer applies (as if it were temporarily deleted until the next load, then re-added). Constraints added using the **-Load_only** option are satisfied by each scan load of a RAM sequential pattern and by each scan load of a clock sequential multiple load pattern. A clock sequential multiple load pattern occurs when you issue the **set_pattern_type** command with the **-Multiple_load** switch on, and the **-Sequential** switch set to 2 or more. Note that you cannot use the **-Load_only** and **-NOCapclock_check** switches at the same time.

Examples

Example 1

The following example creates a user-defined ATPG function and then uses it when creating ATPG pin constraints:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q  
add_atpg_constraints 0 /i$135/q  
add_atpg_constraints 1 and_b_in
```

Example 2

The following example creates a user-defined ATPG function and then uses it when creating ATPG constraints. The ATPG constraints are added to the instances inside core1 only.

```
add_atpg_functions mux_1hotsel SELECT1 -cell imux3sux2 S2 S1 S0
add_atpg_constraints 1 mux_1hotsel -instance core1
```

Example 3

The following example adds ATPG constraints to all occurrences of the specified cell inside the instances with pathnames that begin with “/core1/u19/reg”.

```
add_atpg_constraints 1 -cell imux3sux2 S1 -instance /core1/u19/reg*
```

Example 4

The following example shows how to use wildcards in the second level of hierarchy.

```
add_atpg_constraints 1 -cell imux3sux2 S1 -instance /core1/*/reg*
```

Related Topics

[add_atpg_functions](#)

[delete_atpg_constraints](#)

[report_atpg_constraints](#)

add_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Creates an ATPG function that you can then use when generating user-defined ATPG constraints.

Usage

```
add_atpg_functions function_name type {pin.pathname | gate_id# | existing_function_name  
| {-Cell cell_name {pin_name | -Function type pin_name...}...}...}...
```

Description

Creates an ATPG function that you can then use when generating user-defined ATPG constraints.

You can specify any combination of pin pathnames, gate identification numbers, and previously user-defined functions. You can precede any object with the tilde (~) character to indicate an inverted input with respect to the function. If you specify an input pin pathname, the tool automatically converts it to the output pin of the gate that drives that input pin.

Arguments

- ***function_name***

A required string that specifies the name of the ATPG function you are creating. You can use this *function_name* as an argument to the [add_atpg_constraints](#) command.

- ***type***

A required argument specifying the operation that the function performs on the selected objects. The choices for the *type* argument, from which you can select only one, are as follows:

And — The output of the function is the same as for a standard AND gate.

Or — The output of the function is the same as for a standard OR gate.

Equiv — The output of the function is a high state (1) if all its inputs are at a low state (0), or if all its inputs are at a high state (1). So, the function's output is a low state if there is at least one input at a low state and at least one input at a high state.

Select — The output of the function is a high state (1) if all its inputs are at a low state (0) or if one input is at a high state and the other inputs are at a low state. So, if there are at least two inputs at a high state, the function's output is at a low state.

SELECT1 — The output of the function is a high state (1) if one input is at a high state and the other inputs are at a low state (0). So, the function's output is a low state if there are at least two inputs at a high state or all inputs are at a low state.

The following lists the objects on which the function operates. You can use any number of the argument choices in any order.

- **pin.pathname**

A repeatable string that specifies the pathname to the pin on which you are placing the function. If you specify an input pin name, it is automatically replaced with the output pin of the gate that drives that input pin.

- **gate.ID#**

A repeatable integer that specifies the gate identification number.

- **existing_function_name**

A repeatable string that specifies the name of another function you created with the add_atpg_functions command. The *existing_function_name* argument cannot be the same as any pin name in the design. This string cannot be used with the -Cell option.

- **-Cell cell_name {pin_name} | -Function type pin_name}**

A repeatable switch with a corresponding pair of strings that specify the name of a DFT library cell or pin on that cell. You can repeat the *pin_name* argument if you need to constrain multiple pins or nets on a cell.

The -Function switch is a repeatable switch with corresponding strings that specify the function type and a list of specific pins on cell *cell_name* that are arguments to the function. You can repeat the *pin_name* argument if you need to constrain multiple pins. This switch allows you to define a maximum of two layers of functions with the -Cell switch. Normally, you can define two functions, f1 and f2, and then add a third function, f3, that refers to f1 and f2. However, this is not allowed when you use the -Cell switch.

For information on how this switch is used, see [Example 2](#).

If you use the -Cell option, an ATPG function is placed on every occurrence of that cell within the design.

Examples

Example 1

The following example creates an ATPG function and then uses it in an add_atpg_constraints command:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 1 and_b_in
```

Example 2

The following example generates multiple levels of ATPG functions when using the -Cell switch.

For every instantiation of the OR4 cell, the following function is implemented:

```
[ ( A AND B ) OR ( C OR D ) ]
```

where A, B, C, and D are inputs of the OR4 cell. Normally, you can define two functions, f1 and f2, then add a third function, f3, that refers to f1 and f2. However, when you use the -Cell switch

with the add_atpg_functions command, this is not possible. You can use the -Function switch to define up to two levels of functions using the -Cell switch. The syntax is as follows:

add_atpg_functions top_function or -cell OR4 -function and A B -function or C D

This implemented the previously mentioned function, which can be confirmed with the report_atpg_functions command.

Example 3

The following example assigns the two Q outputs of scan cells /u11 and /u12 to always have the same assigned state after loading.

```
add_atpg_functions CELL_INCLUSIVE equiv /u11/Q /u12/Q
add_atpg_constraints 1 CELL_INCLUSIVE
```

Example 4

The following example causes one input to always be the opposite from the other. The tilde (~) character is the negation symbol.

```
add_atpg_functions CELL_EXCLUSIVE equiv ~ /u71/Q /u72/Q
add_atpg_constraints 1 CELL_EXCLUSIVE
```

Related Topics

[add_atpg_constraints](#)
[delete_atpg_functions](#)
[report_atpg_functions](#)

add_bist_capture_range

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Associates a named ATPG capture procedure with a subset of the BIST patterns.

Usage

add_bist_capture_range *atpg_capture_proc_name start_value stop_value*

Description

Associates a named ATPG capture procedure with a subset of the BIST patterns.

Hybrid TK/LBIST Flow Usage

The add_bist_capture_range command is added once per NCP. Once all the NCPs are applied, the tool restarts from the first NCP.

Additionally, the command is automatically added to the logicBIST fault simulation dofile written out during IP generation. For usage information, refer to “[LogicBIST Fault Simulation and Pattern Creation](#)” in the *Hybrid TK/LBIST Flow User’s Manual*.

Arguments

- ***atpg_capture_proc_name***

A required string that specifies the name of the ATPG capture procedure.

- ***start_value***

A required argument that specifies the beginning of a contiguous range for a group of capture procedures. The tool typically generates the start and stop values in the BIST controller synthesis phase, and you should not change the values.

- ***stop_value***

A required argument that specifies the end of a contiguous range for a group of capture procedures. The tool typically generates the start and stop values in the BIST controller synthesis phase, and you should not change the values.

Examples

This example shows the association of NCPs with capture ranges corresponding to the specification - clk1_twice 34%, clk2_twice 33% and clk3_twice 33%:

```
add_bist_capture_range clk1_twice 0 87
add_bist_capture_range clk2_twice 88 171
add_bist_capture_range clk3_twice 172 255
```

Related Topics

[delete_bist_capture_ranges](#)
[report_bist_capture_ranges](#)
[set_lbist_controller_options](#)

add_black_boxes

Context: all contexts

Mode: setup

Defines instances of Verilog modules or cell library models as black boxes and sets the constrained value on output or bidirectional black box pins.

Usage

```
add_black_boxes {{{-Instances ins_pathnames | -Modules module_names}  
[0 | 1 | X | Z]} [-Pin pinname {0 | 1 | X | Z}]...} | {-Auto [0 | 1 | X | Z]}  
[-FAULT_boundary | -NOFAULT_boundary | -NO_Boundary]}
```

Description

A blackbox is an instance that has no defined model or that you want to treat as though it has no model.

You can blackbox a single instance, or every instance of a particular module or cell library model that you do not want included in the netlist used by the tool. You can also use this command to automatically blackbox instances of nonexistent modules or cell library models.

When reading in a Verilog netlist, a warning message is issued when an instance is found that references a model (Verilog module or cell library model) that is not in the netlist or cell library. A referenced model for which a module or cell library model of the same name is not found in the netlist or cell library is considered to be undefined. You must blackbox instances of undefined models; otherwise, an error message is issued when you attempt to perform certain tasks such as leave Setup mode or enter a create_flat_model command. The message lists the names of referenced models that were not found in the netlist or cell library. You can also list the names using “report_black_boxes -undefined”.

Use add_black_boxes with the -Auto switch to automatically blackbox all instances of models that are considered to be undefined. By default, instances that are automatically blackboxed drive Xs on their outputs. Faults that propagate to the black box inputs are classified as ATPG_untestable (AU). You can use the add_black_boxes command to change the output values.

Issuing multiple add_black_boxes commands in succession, the first with the -Module switch and subsequent ones with -Instance, will globally blackbox all instances of a particular module or cell library model while selectively blackboxing particular instances of the same module or cell library model with slightly different pin values.

Tip

-  For optimal performance, issue all add_black_boxes commands after reading the design and library but before issuing any other commands.
-

In dft -rtl context only, the tool automatically blackboxes modules or instances detected as Synopsys DesignWare modules. The actual blackbox operation is done when the tool transitions from setup to analysis mode using the [set_system_mode](#) command. When the tool exits analysis mode, the tool automatically resets the black box definitions.

Arguments

- **-Instances *ins_pathnames***

A switch and string pair that specifies to blackbox the objects specified by the *ins_pathnames* argument. The *ins_pathnames* argument can be a Tcl list of one or more instances or a collection of one or more instances.

Note

 Blackboxing implemented using the -Instances switch always overrides blackboxing implemented with the -Modules switch.

- **-Modules *module_names***

A switch and string pair that specifies to blackbox every instance of the Verilog module or cell library model specified by the *module_names* argument. The *module_names* argument can be a Tcl list of one or more modules or a collection of one or more modules.

- **-Auto**

A switch that blackboxes any netlist instance that references a model (Verilog module or cell library model) that is not in the netlist or in the cell library.

Note

 To properly blackbox instances for Tessent FastScan MacroTest, use the -Instance or -Module switch. The -Auto switch is not intended for use with Tessent FastScan MacroTest. To ensure success, you must also define at least the module (or entity or cell library model) pins, including names, directions, and width of vectors.

Note

 To prevent unintentional black-boxing of new undefined modules in future executions of your scripts, it is recommended to not use the -auto switch in your dofile.

- 0 | 1 | X | Z

An optional literal that specifies pin tie values. When used with the -Instance or -Module switch, it specifies the tie value for any pin not explicitly defined by the -Pin switch. When used with -Auto, it specifies the tie value for every pin. If you specify no value for this option, then the [set_tied_signals](#) command's value is the default.

- **-Pin *pinname0 | 1 | X | Z***

An optional, repeatable switch, string, and literal that specifies the tie value of a particular pin. Use this switch to explicitly define the tie value for a pin, overriding for that pin the global tie value in effect for the **-Instance** or **-Module** switch.

- **-FAULT_boundary**

A switch that keeps pin pathnames at the boundaries of all blackboxed instances and allows boundary pins to become fault sites. This is the default for `celldefine modules and library models.

- **-NOFAULT_boundary**

A switch that keeps pin pathnames at the boundaries of all blackboxed instances, but does not allow boundary pins to become fault sites. This is the default for blackboxed instances that are not `celldefine modules or library models.

- **-NO_Boundary**

A switch that does not keep pin pathnames at the boundaries of all blackboxed instances and does not allow boundary pins to become fault sites.

Note



Do not include this switch when you use the **add_black_boxes** command to blackbox a macro for Tesson FastScan MacroTest. The Tesson FastScan **macrotest** command must be able to find pin pathnames at the macro boundary.

Examples

Example 1

The following example creates a black box for a module named “core,” with each output driven by a tie0 logic gate. The example then overrides, for the single instance “/core1” of the module “core,” the output value of pin1, so it is driven by a tie1 logic gate instead of a tie0. All other instances of the “core” module still have outputs driven by tie0 logic.

```
add_black_boxes -modules core 0
add_black_boxes -instances /core1 -pin pin1 1
```

Example 2

The following example shows the tool’s warning message for instances of undefined models. The example then creates black boxes for these instances.

add_black_boxes

```
// Warning: Undefined modules were found.
// Before using "set_system_mode" or "create_flat_model", you must either
// define the missing modules using "read_verilog" and/or
// "read_cell_library", or use the following command to treat them as
// black boxes:
// add_black_boxes -modules { \
//   and02 \
//   ao2 \
// }
// You can also use "add_black_boxes -auto" to black box all undefined
// modules but it is recommended that you do not add this command to your
// dofile. Doing so may unintentionally black-box new undefined modules
// in future runs.
```

add_black_boxes -auto**Example 3**

The following example shows two commands that keep the pin pathnames at the boundary of each instance of the blackboxed module named “macro1.” Either command allows Tessent FastScan MacroTest to function properly:

add_black_boxes -modules macro1 -fault_boundary

add_black_boxes -modules macro1 -nofault_boundary

Example 4

The following example creates blackboxes for the collection of modules returned by the get_modules command. The get_modules command returns only those modules that have been marked by the user with the user-defined attribute mark_black_box.

add_black_boxes -modules [get_modules -filter mark_black_box]

Example 5

Similar to Example 4, this example operates on instances that have a user-defined attribute mark_black_box.

add_black_boxes -instances [get_instances -filter mark_black_box]

Example 6

The following example creates blackboxes by assigning the collection of modules returned by the get_modules command to a variable. The add_black_box command then operates on the collection contained in the to_black_box variable.

```
set to_black_box [get_modules -filter mark_black_box]
add_black_boxes -modules $to_black_box
```

Related Topics

[delete_black_boxes](#)

[report_black_boxes](#)

[add_browser_data](#)

Context: all contexts

Mode: setup, analysis

Adds the specified data to the active tab of the DFTVisualizer Browser window.

Usage

```
add_browser_data {Gates | Primitives | {FAults [-All] | -Total | fault_class} |  
    T_coverage | F_coverage | Loss_TC | RELevant_test_coverage  
    | ATpg_effectiveness}  
    [-Tab INstance | Library | Clock]
```

Description

Adds the specified data to the active tab of the DFTVisualizer Browser window.

You can use the *-Tab* switch to specify to add the data column to a tab that is not active. You can specify multiple arguments in one execution of this command.

The `add_browser_data` command opens DFTVisualizer, if the tool is not already open, and then opens the Design Browser window. If the Instance tab of the Browser window is active, the specified data displays for each hierarchical submodule instance listed in the Design Browser window. If data is not available, the column is blank for that instance.

Use the [delete_browser_data](#) command to delete data columns.

Arguments

- **Gates**

Required literal that displays the number of Verilog primitives or Tessent Cell library models in each instance listed in the Instance tab of the Design Browser window.

- **Primitives**

Required literal that displays the number of Tessent Cell library primitives in each instance listed in the Instance tab of the Design Browser window.

- **FAults**

Required literal that displays the total number of detected faults for all instances listed in the Browser window. The total number of detected faults is the sum of all *uncollapsed* faults. This number/sum corresponds to the right-most column of data output by the `report_statistics` command. By default, all faults for each fault class display. This switch is valid in the Instance and Clock tabs of the Design Browser window. Options include:

-Total — Switch that lists the number of total faults for all classes.

fault_class — String that specifies a fault class to display faults for. For a summary of available fault classes, refer to the `report_faults` command.

-All — Switch that specifies to display all faults classes. This option displays the following columns in the Browser window: UO, DS, DI, PT, UU, and AU. This is the default.

- **F_coverage**

Required literal that displays the fault coverage data for each instance. This switch is available in the Instance, Library, and Clock tabs of the Design Browser window.

- **T_coverage**

Required literal that displays the test coverage data for each instance. This switch is available in the Instance, Library, and Clock tabs of the Design Browser window.

- **Loss_TC**

Required literal that displays the undetected faults in an instance displayed as a percentage of the testable faults in the entire design. This switch is available in the Instance, Library, and Clock tabs of the Design Browser window.

- **RELevant_test_coverage**

Required literal that displays the test coverage for each submodule after untestable faults have been added/deleted from test coverage calculations. This switch is available in the Instance and Clock tabs of the Design Browser window.

- **ATpg_effectiveness**

Required literal that displays the ATPG effectiveness data for each instance in the Design Browser window. This switch is available in the Instance and Clock tabs of the Design Browser window.

- **-Tab INstance | Library | Clock**

Optional switch and literal that specify the tab in the Design Browser window to which the column is to be added. By default, the add_browser_data command adds data to the active window. You can use this switch to specify to add the data column to a tab that is not the active tab.

Examples

The following example adds a column displaying the number of primitives in each instance. Because the -Tab argument is not specified, by default, the column is added to the active tab.

add_browser_data primitives

The next example adds a column displaying the number of primitives in each instance, and also adds columns providing all fault coverage statistics for each instance:

add_browser_data primitives faults -all

Related Topics

[delete_browser_data](#)

[add_capture_handling](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the data capturing behavior for the given state element.

Usage

```
add_capture_handling {Old | New | X} {gate_id# | pin_pathname | instance_name |  
{-Cell cell_name}} ... [-SInk | -SOurce]
```

Description

Tip

 This command should rarely be used. For most designs, other than those that require mixed capture handling, you should not use this command but rather use split capture handling. Split capture handling is automatically set if needed when you use the [create_patterns](#) command. Alternatively, you can directly set split capture handling using the [set_split_capture_cycle](#) command.

Specifies the data capturing behavior for the given state element.

After changing the data capture handling for selected state elements with [add_capture_handling](#), you need to issue the [set_capture_handling](#) command to allow the tool to automatically identify the upstream state elements with their associated sinks, and the downstream state elements with the associated sources you defined.

When you use the [add_capture_handling](#) command to change the data capture handling settings, you cannot define source points with the new handling behavior if they propagate to sink points that do not have the new behavior, or to non-clock primary outputs. If the tool has different capture handling behaviors for the same state element, the behavior you define with the [add_capture_handling](#) command overrides the behavior you globally defined with the [set_capture_handling](#) command.

Tessent FastScan and Tessent TestKompress limit the scope of the effect of the capture handling behavior to the circuitry between the source and the sink points. You cannot simulate a newly captured effect past the sink point.

You can change the simulation behavior of RAM models with data hold capability using the [add_capture_handling](#) command. This is useful in cases when it is required to model a RAM which has data hold capability but does not introduce any latency.

Arguments

You must choose one of the following three literals to indicate the data capture handling behavior for the specified state elements:

- **Old**

A literal specifying that the source state elements determine their output values for data capture by using the data that existed prior to the current clock cycle. The tool then passes the data on to the source state element's sink state elements. This option is the default behavior upon invocation of Tesson FastScan and Tesson TestKompress.

- **New**

A literal specifying that the source state elements determine their output values for data capture by using the data from the current clock cycle. The tool then passes the data on to the source state element's sink state elements.

- **X**

A literal specifying that the source state elements use the output values from the current clock cycle for data capture unless the previous values are different from the current values. If the values differ, the source passes unknown (X) values onto the source state element's sink state elements.

The following lists the methods for naming the state elements on which the function operates. You can use any number of the four argument choices, in any order.

- **gate_id#**

A repeatable integer that specifies the gate identification number of the object. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- **pin.pathname**

A repeatable string that specifies the name of a pin within the design. When specifying a pin, the pin must either be an output pin or must connect directly to an internal DFF or memory element.

- **instance_name**

A repeatable string that specifies the name of a RAM instance within the design.

Note



The instance name parameter is only valid for RAMs and FFs.

- **-Cell cell_name**

A repeatable switch and string pair that specifies the name of a cell.

- **-SInk**

An optional switch specifying that the state element you name is a termination point for data capture. This is the default.

- **-S**ource

An optional switch specifying that the state element you name is an origination point for data capture.

Examples

The following example changes the data capture handling of a specific gate and then globally assigns the data capture handling for all C3 and C4 rules:

```
add_capture_handling new 1158 -source
set_capture_handling -te new

// Begin capture handling analysis: LS=OLD, TE=NEW (#C3=1 #C4=1),
// #user_pts=1/0
// Capture handling analysis completed: #sources=1, #int_gates=3,
// #sinks=1, CPU_time=0.03 sec
// Warning: 1 scan source points with incompatible handling
// were identified
```

Related Topics

[delete_capture_handling](#)
[report_capture_handling](#)
[set_capture_handling](#)

add_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Creates a new test in the CDP.

Usage

```
add_cdp_test test_name -pattern pattern_file
```

Description

Creates a new test in the CDP. The test is based on the specified pattern file.

The newly created test allows the user to perform two test operations:

- Execute the pattern file to determine the pass/fail status.
- Execute the pattern file and collect failing data.

Arguments

- *test_name*

A required string that specifies the name of the test.

- **-pattern** *pattern_file*

A required switch and string pair that specifies the directory path for a standalone SVF or STIL test pattern file that will be associated with the test.

Examples

The following example adds a test pattern file named *pat.svf* to the CDP and creates a test named *test1* for the *pat.svf* pattern file.

```
add_cdp_test test1 -pattern ./patterns/pat.svf
```

Refer to “[Executing Non-Tessent IJTAG Instruments](#)” in *Tessent SiliconInsight User’s Manual for Tessent Shell* for a sample usage scenario.

Related Topics

[delete_cdp_test](#)[execute_cdp_test](#)

add_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Constrains scan cells (also non-scan cells for CX, TX, DX, and SX constraints) to a constant value.

Usage

```
add_cell_constraints {-Drc C6} | {{chain_name cell_position} | pin.pathname |
{-CHain chain_name} | {-Clock clock_name [-Type {Latch | Flop}]}}
-Instances instance_name ...} {C0 | C1 | CX | OX | XX | TX | DX | SX | C0DX | C1DX}
[-All_patterns | -Scan_patterns]}
```

Description

Constrains scan cells (also non-scan cells for CX, TX, DX, and SX constraints) to a constant value.

For all contexts, you can identify a particular scan cell either by specifying a scan chain name along with the cell's position in the scan chain, or by specifying an output pin pathname that connects to a scan memory element. The constraint value that you specify is placed at either the output pin or the scan cell MASTER. The “patterns -scan” context additionally provides the -Chain and -Clock switches for identifying groups of scan cells to receive a particular constraint.

The rules checker audits the correctness of the data that defines the constrained scan cells immediately after scan cell identification. The checker identifies all invalid scan cell constraints and an error condition occurs.

In the case of scan cells with improper controllability or observability, rather than rejecting these circuits, you can constrain (or mask) their controllability or observability.

Note

 Adding constraints can lower test coverage.

Arguments

- **-Drc C6 (“patterns -scan” context)**

A switch and literal pair that specifies to add a DX cell constraint to each latch and flip-flop that results in a **C6** violation during DRC. Because this switch could have a big impact on the test coverage, we strongly recommend that you use the “set_simulation_options -C6_mask_races on” command to resolve C6 violations that occur when either the data port is faster than the clock port, or when both ports have almost the same delay and it is indeterminate as to which port is faster. For more description of these issues, see the **-C6_mask_races** argument description for the [set_simulation_options](#) command.

- ***pin.pathname***

A string that specifies the name of one of the following types of pins:

- An output pin of the scan cell.
- An output pin directly connected through only buffers and inverters to the output of a scan memory element. The scan memory element is set to the value that you specify such that the pin is at the constrained value.
- An output pin of a non-scan cell if you are applying a CX, TX, DX, or SX constraint.

Except as noted above, the pin pathname must resolve to a scan memory element. Buffers and inverters may reside between the pin and the memory element.

You can also use an introspection command to generate a Tcl collection of pins. For example:

```
add_cell_constraints [get_gate_pins GO_ID_REG_reg_*/q] OX
```

- ***chain_name cell_position***

A string pair that specifies the name of the scan chain and the position of the cell in the scan chain. The scan chain must be a currently-defined scan chain and the position must be an integer where 0 is the scan cell closest to the scan-out pin. You can determine the position of a cell within a scan chain by using the report_scan_cells command.

The MASTER memory element of the specified scan cell is set to the value that you specify; there is no inversion. However, the tool may invert the output pin of the scan cell if there is anything between it and the MASTER memory element if inversion exists between the MASTER and the scan output pin of the scan cell only.

- **-CChain *chain_name* (“patterns -scan” context only)**

A switch and string pair that places cell constraints on all cells in the specified scan chain, except any that are already constrained; this switch does not override a pre-existing cell constraint on a cell. This switch is valid in non-Setup system modes only.

- **-Clock *clock_name* (“patterns -scan” context only)**

A switch and string pair that adds cell constraints to all latches and flip-flops controlled by the clock, *clock_name*. If the cell that includes the latch or flip-flop is already constrained, this switch does not override the pre-existing constraint. This switch is valid in non-Setup system modes only.

Note

 This switch only supports constraint types OX, DX and SX.

- **-Type Latch | Flop (“patterns -scan” context only)**

An optional switch and literal pair that limits the -Clock switch’s action to adding cell constraints on just the latches or just the flip-flops controlled by the clock, *clock_name*. This switch is valid in non-Setup system modes only.

- **-Instances *instance_name* ...**

An optional switch and string pair that specifies one or more instance names upon which to add cell constraints. This switch supports all types of constraints for scan cells inside the specified instance.

This switch also supports all types of constraints for non-scan cells and transparent latches in the scan path inside the specified instance, except for the following cases:

- If the state element is learned as TIE-0 or TIE-1, you cannot add any cell constraint.
- If the DRC learns the state element with loading value 0 or 1 when switching to capture, you cannot add cell constraints C1 and C1DX to the state element with loading value 0, or add constraints C0 and C0DX to the state element with loading value 1.
- If the DRC analysis classifies the state element as TIE-X during capture, you cannot add C0, C1, C0DX, and C1DX cell constraints to this state element.
- If the DRC analysis classifies the state element as a transparent latch during capture, you cannot add C0, C1, C0DX, or C1DX cell constraints to this state element.
- You cannot add OX cell constraints to non-scan cells or transparent latches in the scan path.

- **-Scan_patterns (“patterns -scan” context only)**

An optional switch that applies the cell constraint only when creating scan test patterns.

- **-All_patterns (“patterns -scan” context only)**

An optional switch that applies the cell constraint when creating all patterns (scan and chain test patterns). Note that this switch is not valid with the -Drc switch in the same command line. This is the default.

Note



“dft -edt” context if you specify many binary constraints, the tool may not be able to create chain test patterns that satisfy all the constraints.

- Constraint Values

C0

A literal that constrains the scan cell to load value 0 only.

C1

A literal that constrains the scan cell to load value 1 only.

CX

A literal that specifies to simulate the loaded scan cell or non-scan value as unknown (uncontrollable).

OX

A literal that specifies to simulate the unloaded scan cell value as unknown (unobservable).

XX

A literal that constrains the scan cell to be both uncontrollable and unobservable.

TX

(“patterns -scan” context only) A literal that constrains the scan cell or non-scan cell to TIE-X during test generation and simulation. Additionally, the logic value X will be loaded into and unloaded from the scan cell.

DX

(“patterns -scan” context only) A literal that specifies for the loaded value to remain until the cell (scan or non-scan) is disturbed by any on clock. When the cell is disturbed, the output value of the cell becomes X for subsequent cycles until the next scan load initializes it to a new value. The cell unloads an X if disturbed, but if not disturbed unloads the loaded value. The tool does not use a scan cell constrained to DX as an observation point.

If you use “`add_cell_constraints -drc c6`”, the tool applies a DX constraint automatically to each latch and flip-flop that results in a C6 violation during DRC. See the description of the “`-Drc C6`” argument for more information.

SX

(“patterns -scan” context only) Similar to DX except that when the cell is disturbed by any on clock, the output value of the cell becomes X *only if the captured value is different* from the loaded (or initialization) value. The tool does not use a scan cell constrained to SX as an observation point.

Note

 If “`set_drc_handling d8 warning`” is in effect for DRC, the tool will automatically apply an SX cell constraint to each D8 failing master cell. This will prevent D8-related simulation mismatches when you verify the patterns in a timing based simulator, but may reduce coverage. If you use “`set_drc_handling d8 note`” for DRC, the tool does not apply SX cell constraints for D8 violations, which ensures test coverage is not affected. But you should use note handling only for cell architectures you know will not produce mismatches.

C0DX

(“patterns -scan” context only) A literal that constrains the scan cell to both load value 0 only and to keep the loaded value until the cell is disturbed by any on clock (C0 and DX). After the clock disturbance, the unload data is X (masked).

C1DX

(“patterns -scan” context only) A literal that constrains the scan cell to both load value 1 only and to keep the loaded value until the cell is disturbed by any on clock (C1 and DX). After the clock disturbance, the unload data is X (masked).

Table 3-3 shows how cell constraints affect the cell during load-capture-unload. The table shows the unloaded (expected) value of the cell given three possible scenarios during capture:

- **Cell Not Disturbed** — All of the cell's clock/set/reset ports are off.
- **Cell Disturbed (same value captured)** — At least one of the cell's clock/set/reset ports are not off (either on or X), and the new stable capture value is the same as the old value in the cell. Because the tools perform a timeless simulation, the cell's new capture value may not be predictable for this scenario (for example, if a timing exception path drives this cell).
- **Cell Disturbed (different value captured)** — At least one of the cell's clock/set/reset ports is not off (either on or X), and the new stable capture value is different from the old value in the cell.

Note that the table uses 'a' and 'b' to represent different Boolean values (for example, if $a=1$, then $b=0$ or X).

Table 3-3. Example Cell Value Changes with Different Constraints for Scan Patterns

Cycle:	Load	Capture			Unload
Constraint	Loaded Value (initialized value)	Cell Not Disturbed	Cell Disturbed (same value captured)	Cell Disturbed (diff. value captured)	Unloaded Value
No constraint	a	a	a	b	same as capture value
C0	0	0	0	b	same as capture value
C1	1	1	1	b	same as capture value
CX	X	X	X	b	same as capture value
OX	a	a	a	b	X
XX	X	X	a	b	X
TX	X	X	X	X	X
DX	a	a	X	X	same as capture value
SX	a	a	a	X	same as capture value
C0DX	0	0	X	X	same as capture value
C1DX	1	1	X	X	same as capture value

Table 3-4 shows, for chain patterns, the different load and unload values of a scan cell with different constraints applied. Notice for chain patterns that CX, XX and TX are equivalent, and DX and SX are the same as no constraint.

Table 3-4. Example Scan Cell Value Changes with Different Constraints for Chain Patterns

Constraint	Loaded Value	Unloaded Value
No constraint	a	a
C0	0	0
C1	1	1
CX	X	X
OX	a	X
XX	X	X
TX	X	X
DX	a	a
SX	a	a
C0DX	0	0
C1DX	1	1

Examples

Example 1

The following example applies one constraint for use during the generation of scan patterns and one constraint for use during the generation of all patterns. Then it displays a list of all the constrained scan cells:

```
add_cell_constraints chain1 5 c0
add_cell_constraints /reg_d2/_Q OX -all_patterns
report_cell_constraints

// constraint chain   cell   cell_instance constraint      pattern scope
//   value     name    position      name      properties
// -----
//   CO        chain1    5           (Dynamic)   Scan Patterns only
//   OX        chain4    7   /reg_d2/_Q       (Dynamic)   Chain + Scan Patt's
```

Example 2

The following example constrains all the remaining scan cells in the chain1 scan chain to a constant one:

```
add_cell_constraints -chain chain1 c1
```

Example 3

The following example solves C6 rule violations resulting from DRC:

```
add_cell_constraints -drc c6
```

Example 4

The following example adds an OX constraint to all flip-flops controlled by clock /clk1:

```
add_cell_constraints -clock /clk1 -type flop OX
```

Related Topics

[delete_cell_constraints](#)
[report_cell_constraints](#)
[report_scan_cells](#)
[report_scan_chains](#)

add_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Specifies DFT library models for user-defined test points, system-generated test points, and system-generated test logic.

Usage

```
add_cell_models dftlib_model {-TYPE {INVerter | AND | BUFFer | OR | NAND | NOR |  
XOR | CLOCK_BUFFer |{MUX mux_select mux_in0 mux_in1}  
| {SCAN_CELL clock data [-POSEDGE_clock | -NEGEDGE_clock]}  
| {DFF clock data [-POSEDGE_clock | -NEGEDGE_clock]}  
| {LATCH enable data [-ACTIVE_HIGH_clock | -ACTIVE_LOW_clock]}
```

Description

Specifies DFT library models for user-defined test points, system-generated test points, and system-generated test logic.

When adding test logic circuitry, multiple models from the DFT library are used. The *cell_type* attribute in the library model description specifies the components available for use as test logic. You can use the *add_cell_models* command instead of defining the *cell_type* attribute in the library model description.

For more information on the design library attribute, see [cell_type](#) in the *Tessent Cell Library Manual*.

Note

 The *add_cell_models* command does not recognize `celldesign cells as library cells and issues an error stating that these cells are not found in the library.

Arguments

- ***dftlib_model***

A required string that specifies the name of a cell model in the DFT library used for test logic, buffer tree, lockup cell, or test point insertion.

- **-TYPE**

A required switch and argument pair that specifies a model type. Options include the following:

INVerter — A literal specifying an inverter cell.

AND — A literal specifying an AND cell.

BUFFer — A literal specifying a buffer cell.

OR — A literal specifying an OR cell.

NAND — A literal specifying a NAND cell.

NOR — A literal specifying a NOR cell.

XOR — A literal specifying an exclusive OR cell.

CLOCK_BUFFer — A literal specifying a clock buffer cell.

MUX *mux_select mux_in0 mux_in1* — A literal and three strings specifying a 2-1 multiplexer and the names of the selector pin and both data pins.

SCAN_CELL *clock data* [-POSEDGE_clock | -NEGEDGE_clock] — A literal and two strings specifying a mux-DFF scan cell with the following input pins: clock, data, scan in, and scan enable

You must specify the names of the clock and data pins only of the DFT library cell model. You may also include an optional switch to specify the inversion on the clock pin. This option works in conjunction with the [add_tied_signals](#) command.

DFF *clock data* [-POSEDGE_clock | -NEGEDGE_clock] — A literal and two strings specifying a D flip-flop with input pins clock and data. You must specify the names of the clock and data pins only of the DFT library cell model. This option works in combination with the [add_tied_signals](#) command. If you are defining this model for use with lockup cells, you may also use an optional switch to specify the overall inversion on the clock pin.

LATCH *enable data* [-ACTIVE_HIGH_clock | -ACTIVE_LOW_clock] — A literal and two strings specifying a D latch with input pins enable and data. You must specify the names of the enable line and the data pin only of the [DFT library cell model](#). If you are defining this model for use with lockup cells, you may also use an optional switch to specify the overall inversion on the clock pin. If you use this command to add an active_low latch, you must specify “`add_cell_models dftlib_model -type latch enable data -active_low_clock`”. The default is -active_high_clock.

Examples

The following example shows a typical use of test logic involving the set, reset, and clock pins on sequential elements (flip-flops). The tool can usually ensure controllability of sequential elements with model types of And, Or, and, Mux.

```
add_clocks 0 clk
set_test_logic -set on -reset on -clock on
set_system_mode analysis
...
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux si a b
```

Related Topics

[delete_cell_models](#)
[report_cell_models](#)
[set_test_logic](#)

add_chain_masks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis
Mode: setup and analysis, when EDT is off
Mode: setup only, when EDT or LBIST is on
Mode: setup only for -Load_value switch
Mode: setup only for -instance -block_chain_index_list switch pair
Mode: setup only for -cell_constraints switch
Masks the load, capture, and/or unload values on the specified scan chains during simulation.

Usage

Available when EDT is Off

```
add_chain_masks chain_name... [-Cell_constraints {OX | XX | TX}] [-Unload_value X]
```

Available when EDT is On

```
add_chain_masks {chain_name... | -INStance instance_name -Block_chain_index_list  
  block_chain_index_list ...} [-Cell_constraints {OX | XX | TX}]  
  [-USed_chains {OFF|ON}] [-Load_value {0 | 1}] [-Unload_value {X | 0 | 1}]
```

LogicBIST Simulation

```
add_chain_masks {chain_name... | -INStance instance_name -Block_chain_index_list  
  block_chain_index_list ...} [-Cell_constraints {OX | XX | TX}]  
  [-USed_chains {OFF|ON}] [-Load_value {0 | 1}] [-Unload_value {0 | 1}]
```

Description

Masks the load, capture, and/or unload values on the specified scan chains during simulation.

The add_chain_masks command allows you to mask specified scan chain values from the test results. This command also allows you to specify a circuit response (0/1) to the compactor.

You can use this command to specify that custom masking logic exists between the scan chain outputs and the compactor (when EDT is ON), between the scan chain outputs and output pins (when EDT is OFF), or on the input of the scan chains. You must create and appropriately control the custom masking logic.

Logic BIST Simulation

Allows you to specify the scan chains that are masked during fault simulation and their load and unload values. This is useful to get around design issues or broken scan chains post manufacturing. If you use the hybrid TK/LBIST controller, then this command is used to specify the scan chains that need to be masked and the constrained load values for those chains. The tool automatically creates the data to be loaded into the static masking register to implement the masking. On the other hand, if you use your own LBIST controller, then this command allows you to specify the chain masking values corresponding to the hardware that you have added.

When you use a single scan chain mask register mask a group of scan chains, the **add_chain_masks** command automatically applies the masking information from the specified masked chain to all other scan chains in the group.

Pattern Generation with EDT On

When using the Hybrid EDT/LBIST controller in EDT mode, and you use a single scan chain mask register mask a group of scan chains, the **add_chain_masks** command automatically applies the masking information from the specified masked chain to all other scan chains in the group.

Pattern Generation with EDT IP TCD and Unused Chains

The EDT TCD flow enables you to specify that a chain is present but inactive and not to be traced. You specify these inactive (unused) chains by using a core instance (using **-Instance**) and the list of chain indices on the EDT controller instead of specifying chain names. You identify the inactive chains with the **-used_chains** switch. The **-used_chains** switch refers to the chain(s) references and is used as follows:

- **-used_chains OFF** — Tracing of the scan chain(s) disabled.
- **-used_chains ON** — Tracing of the scan chain(s) enabled.

If the chain is referenced using a core instance, then specification of **-used_chains** off/on is mandatory.

Arguments

- **chain_name...**
Required, repeatable string that specifies the name of a scan chain in the current design.
- **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list...***
Required set of switches and values that specifies a core instance and the list of chain indices starting with “1” on the EDT controller. When this is used, the masking applied to the specified chain is OX. Use this option in conjunction with the **-USed_chains** switch.
- **-Cell_constraints OX | XX | TX**
Optional switch and literal pair that specifies the cell constraints and observe values for the masking of the specified scan chain(s).
 - OX** — Load and capture values are applied, and the fault observation is not included in the results.
 - XX** — Load and unload values are not applied, and values are captured normally for all scan cells in the scan chain for multiple-cycle tests.
 - TX** — All scan cell values are tied as X.

- **-Load_value 0 | 1**

Optional switch and literal pair that specifies the value that is applied to the (internal) scan chain input. Options are:

0 — Specifies the load value is 0.

1 — Specifies the load value is 1.

For EDT, this command allows you to load a value (0/1) at the input of the specified (internal) scan chains. It is assumed that you insert the hardware that constrains the loading value.

For LBIST, this command is used to specify the value (0/1) to be loaded at the input (internal) of the scan chains. The value loaded should be consistent with the hardware that has been added to constrain the loading value at the scan chain inputs.

If there are cell constraints added on a scan chain and also “add_chain_masks –load_value” is applied on the same chain, the tool issues an error. EDT Finder is turned off automatically with using “-load_value” for EDT.

- **-Unload_value X | 0 | 1**

Optional switch and literal pair that specifies the unload value used to calculate the circuit response. Options include:

X — Specifies the unload value is X. This value is valid with EDT On or Off. This value is not valid for Logic BIST simulation.

0 — Specifies the unload value is 0. This value is valid only for EDT On, and for Logic BIST simulation.

1 — Specifies the unload value is 1. This value is valid only for EDT On, and for Logic BIST simulation.

- **-USed_chains OFF | ON**

Optional switch and literal pair that specifies whether the tool traces chain references. Choose from the following:

○ **-Used_chains OFF** — Tracing of the scan chain(s) disabled.

○ **-Used_chains ON** — Tracing of the scan chain(s) enabled.

The following usage conditions apply:

- If a bypass chain only includes unused chains, then the tool does not add the chain.
- If a bypass chain includes a mix of used and unused chains, the tool issues a warning.
- If all bypass chains have unused chains such that there are no bypass chains left to add, the tool issues an error message.

This switch is only supported with EDT instrument instances.

For further details on using this switch in the TCD flow, refer to the section “[Pattern Generation with EDT IP TCD and Unused Chains](#)” on page 125.

Examples

Example 1

The following example does not apply load values (no controllability) or unload values (no observability) and uses all the cells from *chain1* during a multicycle test. The 0 unload value is used to calculate the circuit response for the external channels.

```
add_chain_masks chain1 -cell_constraints xx -unload_value 0
```

Example 2

The following example shows the add_chain_masks in the pattern -scan context. The command masks the scan chain input by loading a 0, and masks the scan chain output by unloading a 0 on chains chain1 and chain2.

```
add_chain_masks chain1 chain2 -load 0 -unload 0
```

```
report_chain_masks
```

Masked scan chain	Load value	Unload value	Cell constraints
-----	-----	-----	-----
chain1	0	0	OX
chain2	0	0	OX

Related Topics

[delete_chain_masks](#)

[add_scan_chains](#)

[report_chain_masks](#)

[reset_di_faults](#)

add_clocks

Context: all contexts

Mode: setup

Adds scan or non-scan clocks to the clock list for proper scan operation.

Usage

For synchronous source clocks:

```
add_clocks [off_state] port_list [-PULSE_Always | -PULSE_In_capture] [-Label label]
           [-dft_inject_node name] [-no_dft_connections]
```

```
add_clocks [off_state] pin_list [-PULSE_Always | -PULSE_In_capture] [-Label label]
           [-PSEudo_port_name name] [-CAPture_only] [-silent] [-dft_inject_node name]
           [-no_dft_connections]
```

```
add_clocks [off_state] port_list [-FREQ_Multiplier int -PULSE_Always] [-Label label]
           [-dft_inject_node name] [-no_dft_connections]
```

For asynchronous source clocks:

```
add_clocks port_list {-PEriod time [s | ms | us | ns | ps | fs]} [-FREQ_Multiplier int]
           [-Label label] [-dft_inject_node name] [-no_dft_connections]
```

```
add_clocks pin_list {-PEriod time [s | ms | us | ns | ps | fs]} [-PSEudo_port_name name]
           [-FREQ_Multiplier int] [-Label label] [-silent] [-dft_inject_node name]
           [-no_dft_connections]
```

For generated clocks (for ICL extraction and memory BIST pre-DFT DRC) patterns -ijtag and dft (with no sub context):

```
add_clocks pin -REFERENCE {pin | port | pseudo_port}
           [-REFERENCE_Inv {pin | port | pseudo_port}] [-FREQ_Multiplier int]
           [-FREQ_Divider int] [-PSEudo_port_name name] [-Label label] [-dft_inject_node name]
           [-no_dft_connections]
```

For adding a clock branch (for ICL extraction and memory BIST pre-DFT DRC):

```
add_clocks pin [-Branch] [-Label label]
```

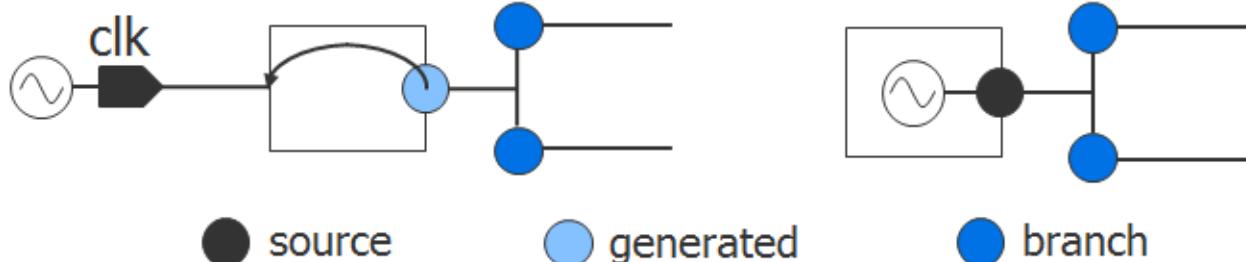
Description

Declares a pin or a port as a clock. The defined clock may be of type source or a generated source. It may also be of type branch. The allowed types of clocks you can define depends on the current context you are using.

A clock is defined as being an asynchronous source when it is defined with a -period option. A clock is defined as generated when it has a -reference option. A clock is defined as a branch when it has the -branch option. Otherwise, the clock is considered a synchronous source. The various clock types are illustrated in [Figure 3-1](#). The source clock represented with the black symbols are shown as asynchronous sources but they could also be synchronous sources. An asynchronous clock is free running and pulses with an arbitrary period. A synchronous clock

pulses or is kept idle under the control of the tester or in the ATPG simulator. The idle state is controlled using the off_state value.

Figure 3-1. Illustration of various clock types



For ATPG and scan DRC, only synchronous clocks are supported. Branch clocks are ignored and generated clocks are converted into synchronous clocks by ignoring the specified -reference, -freq_divider and -freq_multiplier options. You can constrain a clock port to its off-state in order to suppress its use as a capture clock during the ATPG process.

For ICL extraction and memory BIST pre-DFT DRC, all clock types are supported. This enables you to fully describe the complete functional clock network as it pertains to the different instruments and test circuitry as well as by the various memories.

During ICL extraction, clock sources, generated clocks and branch clocks are represented with ICL constructs. Clock sources defined on ports of the current design are reported as ClockPorts with the tessent_clock_periods attribute capturing its period. You will notice that the value of the tessent_clock_periods attribute is “all <period>”. This is to be ready for the time where we will allow a clock to have a different period for different test modes. Today, there is only period described in ICL and it applies to all modes.

```
ClockPort <name> {
    Attribute tessent_clock_periods = "all <real>ns";
}
```

The clocks defined on internal pins with the add_clocks command are captured in ICL using a module named <design_name>_add_clock_<int> as shown here:

```
Module <design_name>_add_clock_<int> {
    // Created by ICL extraction
    Attribute tessent_clock_type = "source|generated|branch";
    ClockPort <input_name>;
    ToClockPort <output_name> {
        Source <input_name>;
        FreqMultiplier <int>;
        FreqDivider <int>;
    }
}
```

Those ICL modules are created only if the internal clock pin is found:

- When ICL Extraction traces from a module that is matched to an ICL module.
- When ICL Extraction traces backwards from a primary output that has been declared as a ToClockPort by means of “`add_icl_ports <portname> -type to_clock`”.
- When ICL Extraction traces from another internal clock pin that has been found before.

In short, those ICL Modules representing the internal clocks are created only if there is something described in ICL to which the internal clock is connected.

The clock label is stored using the attribute “`tessent_clock_domain_labels`” when instantiating the ICL module. The `<pin_name>` value reflects the pin name in the design relative to the current design. This information is useful to generate SDC as the clock pin in the design is identified in the ICL instance.

```
Instance add_clock_inst_<int> Of <design_name>_add_clock_<int> {
    InputPort ClockIn = <source_name>;
    Attribute tessent_clock_domain_labels = "<label_name> {<pin_name>}";
}
```

When issuing the `set_current_design` in the dft context, source or generated clocks modeled in ICL are automatically inferred as `add_clocks`. This is done so that you do not have to declare them manually in order to perform the memory BIST pre-Dft DRC rules [Scan Cell Data Rules \(D Rules\)](#) to [DFT_C5](#). Those clocks are automatically deleted when entering the patterns -ijtag context so that ICL extract sees your real ICL modules. To have your clocks automatically extracted during design elaboration, the ICL module matching the design module must have the syntax illustrated below where a ToClockPort exists with a Source pointing to a ClockPort on the same module. You use the Period element to model internal oscillators and the FreqMultiplier and FreqDivider elements to model PLLs and clock dividers. If your PLLs and clock dividers are programmable, you can easily enable their programming using IJTAG if you follow the description found in the [ProcedureStep](#) section.

```
Module <module_name> {
    ClockPort <input_name>;
    ToClockPort <output_name> {
        Source <input_name>;
        FreqMultiplier <int>;
        FreqDivider <int>;
        Period <int><time_units>;
    }
}
```

If you issue the `report_clocks` command after having run the `set_current_design` command in the dft context, you will see the inferred clocks if any were inferred. They have a name following this format “`inferred_icl_clock<int>`” as shown below. You can use the `report_ijtag_instances` command to see the design instance for which an ICL module was matched. You can use the “`set_design_sources` -format” ICL command to modify the search path for ICL files.

```
User-defined Clock (1):
=====
Generated Clock
=====
-----  -----  -----  -----  -----
Name      Label   Reference   FreqMult   Pin(s)
-----  -----  -----  -----  -----
inferred_icl_clock0  ck0    clk_div/clk_in  1 / 4  clk_div/clk_out
```

Arguments

- *off_state*

An optional literal that specifies the off state of the clock. This option is only used for synchronous clock sources as asynchronous and generated clocks cannot be stopped. The off state of a branch clock is inferred from the off state of the clock in its fanin. The choices are:

0 — A literal specifying the off-state value is 0 (default value).

1 — A literal specifying the off-state value is 1.

Note

 The tool ignores the off_state value when specified for asynchronous and generated clocks.

- *pin_list*

A repeatable string that lists the internal pins to define as clocks. The strings can include any number of asterisk (*) and/or question mark (?) wildcard characters and they must match pin names in the current design.

- *port_list*

A repeatable string that lists the primary input ports to assign as clocks. This string can include any number of asterisk (*) and/or question mark (?) wildcard characters and they must match port names on the current design.

- *-PSEudo_port_name name*

An optional switch and string pair that specifies the name for a new primary input clock pin that drives all of the internal pins specified with the pin_list argument. The name argument must be a simple name that begins with an alpha character and can contain only alphanumeric characters and underscore characters ('_'). Wildcards are not allowed. In ATPG application, you can use this option to map a list of internal clock pins to a single pseudo port. In all other application, the pin_list can only include one pin name when specifying the -pseudo_port_name option so that each internal clock is associated with a unique pseudo port name.

- *-CAPture_only*

An optional switch that specifies that the corresponding pseudo port is only treated as a source clock during capture. For test procedures other than capture, the tool treats this clock

pin as an internal pin, therefore its simulation value will be decided by its driver in the netlist.

- **-PULSE_Always**

An optional switch that identifies the specified pin as a pulse-always clock. A pulse-always clock is pulsed during every cycle of the pattern set.

During ATPG, the clocks you specify as pulse-always are allowed to also pulse with any other clock, ignoring the specified clock restriction setting defined by the [set_clock_restriction](#) command. Pulse-always clocks are automatically pulsed in the external mode of named capture procedures (if present). For more information on named capture procedures, see “[Capture Procedures \(Optional\)](#)” in the *Tessent Shell User’s Manual*. Unless you define an asynchronous free-running clock, you must specify the clock in the internal mode of named capture procedures in order to simulate the effect of the clock pulses.

An [Always Block](#) is automatically created in the procedure file if it does not already exist, and the pulse-always clock is pulsed in this block.

- **-PULSE_In_capture**

An optional switch that identifies the specified pin as a pulse-in-capture clock. This information is only used during ATPG and ignored in all other applications. A pulse-in-capture clock is pulsed in every cycle of the capture window. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting defined using the [set_clock_restriction](#) command.

- **-PEriod time_value [s | ms | us | ns | ps | fs]**

An optional switch and real number pair that specifies the period of a free-running asynchronous clock. The time_value is a real number followed by an optional time unit that defaults to “ns.” Note that ATPG and scan DRC treat asynchronous free-running clocks as X values during the capture cycles.

The period of an asynchronous free-running clock affects the timescale and precision of the Verilog test bench. For more information, refer to the description of the “[set time scale](#)” statement in the *Tessent Shell User’s Manual*.

- **-Label label**

An optional switch and string pair that specifies an arbitrary label for a specified clock. This switch is not valid if the pin_list or port_list values have more than one element. The label name must be a single string that begins with a letter and consists only of letters, numbers, and underscores. Wildcards are not allowed.

Note



The clock label that can optionally be specified with the `add_clocks` command only works with the following commands: `get_clocks`, `get_clock_option`, `set_clock_options`, and `delete_clocks`. For all other commands, you must still use the actual clock pin, port, pseudo-port name, or collection returned by the `get_clocks` command.

If you do not specify a label, the tool automatically assigns a label using the following algorithm:

```

if clock is a port
    if clock name begins with "\" (escaped)
        go to "Choose CK# label step"
    else if port is a bus
        take the scalar name followed with _# corresponding to the bit
        index as the label name
        if the label name does not pass the uniqueness check
            go to "Choose CK# label step"
    else if the port name begins with [a-z] [A-Z], and only contains
        letters, numbers and _,
        if it passes the uniqueness check
            use the port name as the clock label name
        else
            go to "Choose CK# label step"
    If the clock is on a pseudo port
        if the port name begins with [a-z] [A-Z], and only contains
            letters, numbers and _,
            if it passes the uniqueness check
                use the port name as the clock label name
            else
                go to "Choose CK# label step"

Choose CK# label step:
    Take ck#, where # is chosen as the next available integer such
    that
        "ck#" is not an already used clock label and is not an existing
    top-
        level port name or an existing pseudo port name.
    The only time the label "CK#" is allowed to be used when a port or
    a pseudo port by the same name already exists is when the clock is
    defined on that port or that pseudo port itself.

```

- **-dft_inject_node *dft_inject_node***

An optional switch and string pair that specifies where the DFT object affecting the clock (for example, a TCK injection multiplier) should be inserted along the clock path. You can introspect the status of this switch with the “[get_clock_option -dft_inject_node](#)” command and switch.

- **-no_dft_connections**

A Boolean switch that instructs the tool to not infer DFT connections to this clock. When choosing the clock for a dedicated wrapper cell or a test_point in the [insert_test_logic](#) command, the clock used by the majority of the flip-flops in the fanin or fanout of the node is normally used. If that clock was added with the -no_dft_connections switch, it is not chosen and instead the next majority clock is used. If all clocks the fanin or fanout were added with the -no_dft_connections switch, the largest clock domains in the design without this switch are selected.

- **-FREQ_Multiplier *int***

An optional switch and integer pair that specifies an integer to reflect the multiplication factor the clock has with respect to its specified -reference value.

- **-FREQ_Divider *int***
An optional switch and integer pair that specifies an integer to reflect the division ratio the clock has with respect to its specified -reference value. The option can also be used with the -period option to define how a clock frequency is divided from a base period. The tool adds special logic in the Verilog test bench to generate all clocks that have the same base period and make sure they remain coherent during simulation.
- **-REFERENCE {*pin | port | pseudo_port*} (“dft (no sub-context)” and “patterns -ijtag” only)**
A switch and string pair that specifies the clock that is the reference of the clock. The multiplication and division relationship between the clock and its reference is described using the -freq_multiplier and -freq_divider options.
- **-REFERENCE_Inv {*pin | port | pseudo_port*} (“dft (no sub-context)” and “patterns -ijtag” only)**
An optional switch and string pair that specifies the second primary input port if the reference pin is a differential pin.
- **-Branch**
An optional switch that defines a pin as a clock branch. You cannot use this switch with any other switches.
A clock branch shares a source with one or more clocks but it is not skew-balanced with the other clocks that share the common source. The -branch switch is useful for describing branches in a clock tree that are balanced within themselves but not with the other branches of the tree. See [Figure 3-1](#) for a graphical representation of a clock branch.
- **-silent**
An optional switch that suppresses the transcription of notes the tool issues.

Examples

Example 1

The following example adds two external pulse-always clocks with off states of 1:

```
SETUP> add_clocks 1 I2 I3 -pulse_always
```

```
SETUP> report_clocks
```

User-defined Clocks (2) :			
Sync and Async Clocks			
=====			
Name	Off State	Constraints	Internal
-----	-----	-----	-----
'I2'	1	Pulse always	No
'I3'	1	Pulse always	No

Note that the report_clocks command does not report a label name because, in the case of ports, the tool generates a label name that is identical to the port name.

Example 2

The following example adds an internal clock with an off state of 0. The `add_clocks` command automatically creates a PI at the new clock pin.

SETUP> add_clocks 0 u4/u2/CLK

```
// Warning: Primary input 'u4/I6' is added at pin '/u4/u2/CLK'
```

SETUP> report_clocks

User-defined Clock (1):

Sync and Async Source Clock						
Name	Label	Off State	Constraints	Internal	Pin(s)	
u4/I6	ck0	0		Yes	u4/u2/CLK	

SETUP> report_primary_inputs

```
USER:      '/u4/I6' (internal pin)
SYSTEM:    '/I1'
...
```

Note that the `report_clocks` command reports the label name that the tool automatically generates for internal pins.

Example 3

The following example adds two internal clocks and merges them together under a `pseudo_port` name of `port_1`:

SETUP> add_clocks 0 u4/u2/CLK u5/u2/CLK -pseudo_port_name port_1

```
// Note: Primary input 'port_1' is added, merging 2 pins.
```

SETUP> report_clocks

User-defined Clock (1):

Sync and Async Source Clock						
Name	Off State	Constraints	Internal	Other Properties	Pin(s)	
'port_1'	0		Yes	Merged internal pin	u4/u2/CLK u5/u2/CLK	

Example 4

The following example adds an external pulse-in-capture clock with a label name:

SETUP> add_clocks 0 I2 -pulse_in_capture -label clock_2

SETUP> report_clocks

```
User-defined Clock (1) :
```

Sync and Async Source Clock				
Name	Label	Off State	Constraints	Internal
'I2'	clock_2	0	Pulse in capture	No

Example 5

The following example specifies two asynchronous free-running clocks that are coherent. The overall period for the coherent group is 1000ns. The clock “aclk” pulses 20 times during that period while “bclk” pulses 30 times during that period, both with a 50% duty cycle:

```
SETUP> add_clocks aclk -period 1000 ns -freq_multiplier 20
```

```
SETUP> add_clocks bclk -period 1000 ns -freq_multiplier 30
```

```
SETUP> report_clocks
```

```
User-defined Clocks (2) :
```

Sync and Async Source Clocks					
Name	Off State	Constraints	Internal	Period	Freq. Multiplication
'aclk'	-	Asynchronous	No	1000.00ns	20
'bclk'	-	Asynchronous	No	1000.00ns	30

The annotations and comments to document the asynchronous free-running clocks in the pattern files occur in the header section of the pattern file right after the block of comments/annotations that are labeled “Begin_Verify_Section.” The annotations are grouped in a block with a begin and end annotation. Here is an example of the WGL block:

```
{ Begin_async_clocks }
{   aclk = period: 1000.00ns, divider: 20 }
{   bclk = period: 1000.00ns, divider: 30 }
{ End_async_clocks }
```

All other pattern formats use the identical block structure and syntax within the annotation or comment. The only differences are using the correct annotation or comment punctuation or statements surrounding this block.

Example 6

The following example works in “patterns -ijtag” only. The following commands produce an internal half-speed clock on the ICL module “chip” during ICL extraction:

```
set_design_sources -format icl -y icl_data
set_current_design chip
add_clocks { pll_block/clock1 } -reference clock1 -freq_divider 2
set_system_mode analysis
```

The resulting ICL created by the ICL extraction looks as follows:

```
Module chip {
    // Created by ICL extraction
    ClockPort clock1;
    ...
    Instance block1_I1 Of block1 {
        InputPort ClkA = add_clock_inst_0.ClockOut;
        ...
    }
    Instance add_clock_inst_0 Of chip_add_clock_0 {
        InputPort ClockIn = clock1;
    }
}

// instanced as chip.add_clock_inst_0
Module chip_add_clock_0 {
    // Created by ICL extraction
    ClockPort ClockIn;
    ToClockPort ClockOut {
        FreqDivider 2;
        Source ClockIn;
    }
}
```

Related Topics

[analyze_control_signals](#)
[delete_clocks](#)
[get_clocks](#)
[get_clock_option](#)
[report_clocks](#)
[set_clock_options](#)
[set_clock_restriction](#)

add_config_element

Context: unspecified, all contexts

Mode: all modes

Adds a configuration element in the configuration data.

Usage

```
add_config_element [name] [-in_wrapper wrapper_object] [-copy_from config_object_spec]
    [-before config_object_spec | -after config_object_spec | -first | -last]
    [-type wrapper | data_wrapper | csv_wrapper | property] [-value value_list]
    [-replace]
```

Description

Adds a configuration element in the configuration tree. The command returns a single element collection that contains the newly-created element.

The behavior of the command is different when adding an element inside a wrapper that fully defines the allowed child elements versus one which allows arbitrary elements. The allowed content of a wrapper is controlled by its metadata description. Wrappers of type single and simple properties exist automatically as soon as their parent wrapper is added. Therefore, these elements cannot be added using this command as they are automatically added when their parents are added. You use the [set_config_value](#) to specify their value.

You can add elements inside a parent wrapper that does not constrain its content (i.e. its metadata has the property AllowUnknown set to on), but you need to add everything inside it and define the type of object using the -type property. See [Example 2](#) for an example that creates a configuration tree with no metadata.

Arguments

- *name*

An optional string that specifies the name of the configuration element being added. When using the -in_wrapper option, the name is the relative pathname with respect to the specified wrapper. When not using the -in_wrapper option, the name is the complete pathname with respect to the root of the partition. The parent wrapper of the element must exist before the element can be added to it. For example DftSpecification(modal,rtl) must exist to add the MemoryBist wrapper into it using the “add_config_element DftSpecification(modal,rtl)/MemoryBist” command. The added element must not already exist unless the -replace option is used, in which case the existing element will be deleted before being added.

- -in_wrapper *wrapper_object_spec*

An option value pair that specifies that the configuration element is to be added inside a specific wrapper. When the -in_wrapper option is specified, the name option defines a name relative to the specified destination wrapper. The *wrapper_object_spec* can be the complete

name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by [get_config_elements](#) or “[get_config_value -object](#)”.

The default partition defines wrappers by the name of tmp(id) which allow unknown elements to be added inside them. If you want to read an arbitrary configuration wrapper into the tool that does not match any defined syntax, you can use the “add_config_element tmp(id)” command to create an empty tmp wrapper. The id can be any arbitrary label and you can create as many tmp wrappers as you need. Once created, you can then use the “read_config_data file -in_wrapper tmp(id)” command to load arbitrary configuration data inside the tmp(id) wrapper. See [Example 2](#) for an example of this usage.

- **-copy_from config_object_spec**

An optional value pair used to specify that the added element is a copy of another one. When the allowed content of the destination parent wrapper is defined with [Wrapper](#), the metadata definition of the source and destination element must be identical (i.e. both definitions must reference the same metadata definition). If this is not the case, an error message is generated. The name of the added element can be omitted when using both the -copy_from and the -in_wrapper options. In this case, the parent wrapper of the copied element and the destination wrapper must be different. The element is added into the destination wrapper using the leaf_name of the copied object when the name option is not specified.

- **-before config_object_spec**

An optional value pair that specifies a configuration element in front of which to add the new configuration element. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object. Also, the name option must be a leaf_name with no hierarchical component.

When the *config_object_spec* value is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. If the -in_wrapper option is not specified, it is a name relative to the root of the partition.

The -before option is mutually exclusive to the -after, -first, and -last options. When none of these four options is specified, -last is assumed.

- **-after config_object_spec**

An optional value pair that specifies a configuration element after which to add the new configuration element. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object. Also, the name option must be a leaf_name with no hierarchical component.

When the *config_object_spec* value is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. It is a name relative to the root of the partition when the -in_wrapper option is not specified.

The -after option is mutually exclusive to the -before, -first, and -last options. When none of these four options is specified, -last is assumed.

- **-first**
An optional switch that specifies to add the new configuration element before any other element in the parent wrapper.
The **-first** option is mutually exclusive to the **-before**, **-after**, and **-last** options. When none of these four options is specified, **-last** is assumed.
- **-last**
An optional switch that specifies to add the new configuration element after all other elements in the parent wrapper.
The **-first** option is mutually exclusive to the **-before**, **-after**, and **-first** options. When none of these four options is specified, **-last** is assumed.
- **-type wrapper | data_wrapper | csv_wrapper | property**
An optional value pair that is only needed and allowed when adding a configuration element for which no [Wrapper](#) exists. When adding an element for which metadata exists, the **-type** option is not allowed because the type of the object is specified in the metadata. When adding an element for which no metadata exists, the **-type** value defaults to “wrapper” when the **-value** option is not specified and to “property” when the **-value** option is specified.
See [Example 2](#) for an example of its usage when adding an element without metadata.
- **-value value_list**
An optional switch and value pair that is used when adding a configuration element that needs a value. The **-value** option is only used when adding a repeatable property such as the connection property in the [Tdr/DataOutPorts](#) wrapper. The supplied value needs to be a well-formatted Tcl list so as to set a value that contains white spaces and is similar to the following:

```
-value [list "a b c"]
```
- **-replace**
An optional switch that suppresses the error that is normally generated when adding a configuration element with a name that conflicts with an existing element. The error only occurs if the element is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

Example 1

The following example creates a DftSpecification using the `add_config_element` command. The generated configuration data is then reported.

```

set_transcript_style input_only
set spec [add_config_element DftSpecification(modal,ex1)]
set ijt [get_config_elements IjtagNetwork -in_wrappers $spec]
set host [add_config_element HostScanInterface(ijt) -in $ijt]
set mbist [add_config_element MemoryBist -in $spec]
set_config_value ijt_host_node -in $mbist HostScanInterface(ijt)
set ctl [add_config_element Controller(c1) -in $mbist]
set step [add_config_element Step -in $ctl]
set mem_int [add_config_element MemoryInterface(m1) -in $step]
set_config_value memory_library_name -in $mem_int ram12x4
set mem_int [add_config_element ReusedMemoryInterface(m2) -in $step]
set_config_value reused_interface_id -in $mem_int m1
set step [add_config_element Step -in $ctl]
set mem_int [add_config_element MemoryInterface(m3) -in $step]
set_config_value memory_library_name -in $mem_int ram12x8
set mem_int [add_config_element ReusedMemoryInterface(m4) -in $step]
set_config_value reused_interface_id -in $mem_int m3
report_config_data DftSpecification(modal,ex1)

DftSpecification(modal,ex1) {
    IjtagNetwork {
        HostScanInterface(ijt) {
            }
        }
    MemoryBist {
        ijt_host_node : HostScanInterface(ijt);
        Controller(c1) {
            Step {
                MemoryInterface(m1) {
                    memory_library_name : ram12x4;
                }
                ReusedMemoryInterface(m2) {
                    reused_interface_id : m1;
                }
            }
            Step {
                MemoryInterface(m3) {
                    memory_library_name : ram12x8;
                }
                ReusedMemoryInterface(m4) {
                    reused_interface_id : m3;
                }
            }
        }
    }
}

```

Example 2

The following examples adds a tmp wrapper which allows anything inside it to show how arbitrary configuration data can be created. Notice the use of the -type options when adding a data row and a csv row. The -type option is not needed when adding a wrapper or a property because it automatically defaults to wrapper or property based on the presence of the -value option.

```
> set tmp1 [add_config_element tmp(1)]set n_wpr [add_config_element
NormalWrapperExmaple(abc) -in $tmp1]
add_config_element -in $n_wpr prop_example -value [list a b c]
set d_wpr [add_config_element DataWrapperExample -in $n_wpr]
# -type is needed otherwise it would default to property as
# -value is not specified
add_config_element -in $d_wpr element1 -type property
add_config_element -in $d_wpr element2 -value 32ns
set c_wpr [add_config_element CsvWrapperExample -in $n_wpr]
add_config_element -in $c_wpr -value [list a b c]
add_config_element -in $c_wpr -value [list d e]
add_config_element -in $c_wpr -value [list f g h] -after <0>
report_config_data $tmp1

tmp(1) {
    NormalWrapperExmaple(abc) {
        prop_example : a, b, c;
        DataWrapperExample {
            element1;
            element2 : 32ns;
        }
        CsvWrapperExample {
            a, b, c;
            f, g, h;
            d, e;
        }
    }
}
```

Related Topics

[delete_config_element](#)
[get_config_elements](#)
[move_config_element](#)

add_config_message

Context: all contexts

Mode: all modes

Adds error, warning, or information messages to configuration elements.

Usage

```
add_config_message -config_object config_object message  
      [-error | -warning | -info | -note] [-display]
```

Description

Adds error, warning, or information messages to configuration elements.

Arguments

- **-config_object config_object**

A required value that specifies the configuration object to which to add the message specified by the message string. The config_object value is the name of a configuration element or a collection containing a single configuration element object.

- **message**

A required string value that specifies the text being added.

- **-error**

An optional Boolean option that specifies that the messages to be added are of type error.

- **-warning**

An optional Boolean option that specifies that the messages to be added are of type warning.

- **-info**

An optional Boolean option that specifies that the messages to be added are of type info.

- **-note**

An optional Boolean option that specifies that the messages to be added are of type note.

- **-display**

An optional switch that specifies to also display the error, warning, or information messages in the transcript. Without this option, the message is silently added to the configuration object.

If the **file_path_name** and **line_number** attributes are set on the configuration object, then the message displays the pathname to the file and the line number in the file where the configuration element is defined. For example:

```
// Warning: /Core(SYNC_1RW_512x64x2)/Memory/OperationSet  
//           Read from file: ./SYNC_1RW_512x64x2.lib, line number: 8
```

Examples

Example 1

The following example adds an error message to a property element. The -display option specifies to also report the error message in the transcript window.

```
add_config_message -config_object /tmp(1)/propertyA -error -display \
"The value A is inconsistent with the design level '[get_design_level]'."
```

Example 2

The following example illustrates the differences between -error, -warning, -info and -note as to how the messages are displayed in the transcript window:

```
// Error: /Core(SYNC_1R1W_16x8)/Memory
    The memory uses a checkerboard test algorithm SMArchCHKBvcd and has no
    column address bits. Memories with checkerboard algorithm and no column
    address bits must have BitGrouping property set to the data word size.
    The BitGrouping value of this memory is changed to 8.

// Warning: /Core(SYNC_1R1W_16x8)/Memory
    The memory uses a checkerboard test algorithm SMArchCHKBvcd and has no
    column address bits. Memories with checkerboard algorithm and no
    column address bits must have BitGrouping property set to the data
    word size. The BitGrouping value of this memory is changed to 8.

// Core(SYNC_1R1W_16x8)/Memory
    The memory uses a checkerboard test algorithm SMArchCHKBvcd and has no
    column address bits. Memories with checkerboard algorithm and no column
    address bits must have BitGrouping property set to the data word size.
    The BitGrouping value of this memory is changed to 8.

// Note: /Core(SYNC_1R1W_16x8)/Memory
    The memory uses a checkerboard test algorithm SMArchCHKBvcd and has no
    column address bits. Memories with checkerboard algorithm and no column
    address bits must have BitGrouping property set to the data word size.
    The BitGrouping value of this memory is changed to 8.
```

Related Topics

[delete_config_messages](#)
[report_config_messages](#)
[get_config_messages](#)

[add_config_tab](#)

Context: all contexts

Mode: all modes

Adds one configuration tree tab to the DFTVisualizer Configuration Data window.

Usage

`add_config_tab wrapper`

Description

Adds one configuration tree tab to the DFTVisualizer Configuration Data window.

Arguments

- *wrapper*

An value that specifies the name of the configuration wrapper. The value is either the name of a wrapper or a collection that contains exactly one configuration wrapper. The wrapper is the root wrapper displayed in the tab.

Examples

This example displays a tcd_memory wrapper in the Configuration Data window. Notice the use of the get_config_element command to obtain the configuration object. The name could not be used directly in the add_config_tab because the tool only looks in the default partition. The display_specification command displays a DftSpecification in a tab of the Configuration Data window.

`add_config_tab [get_config_element Core(RAM12x8) -partition tcd]`

Related Topics

[delete_config_tabs](#)

[display_specification](#)

add_control_points

Context: dft -test_points

Mode: setup, analysis

Specifies user-defined control points during test point insertion.

Usage

```
add_control_points -location pin/port_spec -type {AND | OR} [-clock clock_pin/port]  
[-enable enable_pin/port]
```

Description

Specifies user-defined control points during test point insertion.

Arguments

- **-location *pin/port_spec***

A required switch and string that specifies the pathname to the control point location.

The *pin/port_spec* can be a Tcl list of one or more pins/ports or a collection of one or more pin/port objects.

- **-type {AND | OR}**

A required switch and literal pair that specifies the gate type to be inserted at the control point location.

- **-clock *pin/port_spec***

An optional switch and string pair that specifies the path name of the clock signal used to clock the control point. This can be a collection or list, but should have only one element.

- **-enable *pin/port_spec***

An optional switch and string pair that specifies the path name of an existing net used to enable/disable the control point. This can be a collection or list, but should have only one element.

Examples

Example 1

The following example adds a type OR control point and a type AND control point. For the AND control point it also specifies the path name of the clock signal used to clock the control point:

```
add_control_point -location gc[1] -type OR  
add_control_point -location gd[1] -type AND -clock gclk4  
report_test_points
```

Example 2

The following example specifies the path name of an existing net that is to be used to enable/disable the control points. For the AND control point it also specifies the path name of the clock signal used to clock the control point. Then it proceeds to insert the control points into the netlist:

```
set_test_point_insertion_options -control_point_enable cp_enable
add_control_point -location ga[1] -type OR -clock buf1/Z
report_test_points
...
set_system_mode analysis
...
insert_test_logic
```

Related Topics

[add_observe_points](#)
[report_test_points](#)
[delete_test_points](#)

add_core_instances

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Adds a core instance to the design by associating/binding the core description currently in memory with the specified core instance(s) in the design.

Usage

```
add_core_instances [-core core_name] [-mode mode_name]
  {-modules module_objects | -instances instance_objects | -current_design}
  [-design_id design_id] [-parameter_values {parameter_list}] [-silent]
```

Description

Adds a core instance to the design by associating/binding the core description currently in memory with the specified core instance(s) in the design. If the core description is not currently in memory, you can read it in using the [read_core_descriptions](#) command. You can report the available core descriptions by using the [report_core_descriptions](#) command.

The [read_core_descriptions](#) command must have already been executed before using this command.

You can specify the core(s) with which to associate the core description by providing either absolute instance pathnames to the cores or unqualified module names, or by using the *-current_design* switch.

Arguments

- **-core *core_name***

An optional switch and Tcl list that specifies the core name of the core description from which the instantiation should take place. If the *-core* switch is not specified, the tool attempts to match a core description against the module name of the specified module(s) or instance(s). If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool's default module matching options which are consistent with what synthesis tools typically use. For information on the default module matching options and how they can be changed, see the description for the [set_module_matching_options -suffix_pattern_list](#) option.

If the tool is unable to match a specified module or instance, it issues an error with a reminder that the user can explicitly specify which core description to use with the *-core* switch.

- **-mode *mode_name***

An optional switch that specifies the mode of the core. This switch is required if the core has multiple modes.

- **-modules *module_objects***

A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects. The core instances are created for each instantiation of those objects in the design.

- **-instances *instance_objects***

A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects. The core instances are created for each instance in the list.

- **-current_design**

A required switch that specifies to add the core instance with respect to the current design. This switch is mutually exclusive with the -modules and -instances switches. The mode of the core instance being added by the -mode switch must be different than the mode of the current design specified using the [set_current_mode](#) command.

- **-design_id *design_id***

An optional switch and string pair that specifies searching any opened [Tessent Shell Data Base \(TSDB\)](#) for Tessent Core Descriptions that match the *design_id* value.

If you omit the -design_id switch and string pair and the added instance has the [design_id attribute](#) set, then the tool performs the matching based on the design_id attribute value. The design_id attribute is inherited by instance objects from their modules. Additionally, when you use “[read_design -design_id *design_id*](#)”, the *design_id* attribute is set automatically for these modules and instances.

If you do not set the *design_id* and there is only one TCD in the opened TSDBs that matches the core name, the tool loads the *design_id* in memory and adds the core instance. If the TCD is already loaded into memory the -design_id switch has no effect and the matching is preformed based on the core and module name.

- **-parameter_values {*parameter_list*}**

An optional switch and parameter list that specifies instrument configuration parameters and their values. The parameter list is one or more parameter value pairs. For example:

```
SETUP> add_core_instances -core_cpu_edt -modules cpu_edt \
          -parameter_values {edt_bypass off}
```

The list includes the subset of those parameters that are configurable in the specified instrument. [Table 3-5](#) shows the core parameters and their instruments.

The parameters for core instances created with this command can be reported with the [report_core_instance_parameters](#) command. Their parameters may be modified with the [set_core_instance_parameters](#) command.

Table 3-5. Core Instance Parameters and Values by Instrument

Parameter	Value	Description
Instrument Type: EDT		

Table 3-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
edt_bypass	on off The default is off.	Defines whether the EDT is used in bypass mode. See “ Compression Bypass Logic ” for more information on the parameter and bypass in general.
edt_configuration	A string that is the name of the configuration specified during IP creation. When used with the DFTSpecification flow and the HighCompressionConfiguration wrapper, there are only two valid values: <u>low_compression</u> <u>high_compression</u> The default is low_compression.	Defines which compression configuration is used by the EDT. See “ Dual Compression Configurations ” in the <i>Tessent TestKompress User’s Manual</i> for more information on compression configurations.
edt_low_power_shift_en	on off The default value is based on the EDT IP generation time user-defined power controller status.	Defines whether the EDT is used in low power shift mode. See the “ Power Controller Logic ” section in the <i>Tessent TestKompress User’s Manual</i> for more information on edt_low_power_shift_en.
edt_single_bypass_chain	on off The default is off.	Defines whether the EDT is used in single bypass mode. See “ Dual Bypass Configurations ” in the <i>Tessent TestKompress User’s Manual</i> for more information on edt_single_bypass_chain.
used_input_channels	integer The default value is the maximum available input channels.	Defines how many input channels should be used by the EDT. See the description for used_input_channels in the set_edt_options command arguments in the <i>Tessent Shell Reference Manual</i> .

Table 3-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
tessent_chain_masking	on off The default is off.	Defines if Hybrid TK/LBIST chain masking should be used. See “ Pattern Generation with Internal Chain Masking Hardware ” in the <i>Tessent TestKompress User’s Manual</i> for more information.
Instrument Type: OCC		
capture_window_size <i>integer</i>	integer The default value is the maximum size supported by the OCC instance. For transition patterns, the default value is 2.	Specifies the maximum number of clock pulses during capture cycle. This value must not exceed the registers created during IP creation. By default, the tool creates an OCC able to pulse up to three times between scan loads, but you may not need to use them all. Set this parameter to a value smaller than default to reduce the number of OCC cells and therefore the number of bits that need to be encoded during pattern generation. When the OCC chains are driven by EDT, you free up encoding capacity to be used for the generated tests and may reduce pattern count.
fast_capture_mode	on off The default is off.	Defines whether the fast capture clock is used during capture. See “ Clock Control Operation Modes ” in the <i>Tessent Scan and ATPG User’s Manual</i> for information on this parameter.

Table 3-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
parent_mode	on off The default is off.	Defines whether the OCC is used in parent mode. When the default off is used, the OCC is used in standard mode. See “ The Parent OCC ” section in the <i>Tessent Scan and ATPG User’s Manual</i> for more information on this parameter.
Instrument Type: LPCT		
reset_control	on off The default is off.	Controls the value being loaded into the LPCT Type-3 reset control cell. See “ Type 3 - LPCT Controller-Generated Scan Enable ” in the <i>Tessent TestKompress User’s Manual</i> .
scan_en_control	on off The default is off.	Controls the value being loaded into the LPCT Type-3 scan enable control cell. See “ Type 3 - LPCT Controller-Generated Scan Enable ” in the <i>Tessent TestKompress User’s Manual</i> .

- -silent

An optional switch that specifies to suppress warning messages.

Examples

Example 1

The “[Retargeting Example](#)” in the *Tessent Scan and ATPG User’s Manual* demonstrates the use of this command in a design context.

Example 2

The following example demonstrates the use of the -current_design switch to merge two modes of a core; in this case, the “core” is the top level. In the example, patterns are generated separately for two sub-blocks of a design from the top-level pins. Sub-Block1 is tested as mode M1 of the top level, and Sub-Block2 is tested as mode M2 of the top level. The result, shown in [Figure 3-3](#) on page 155, is the merging of the two pattern sets originally generated for each individual mode into a single pattern set applied at the top level.

The first dofile reads in the top level design and writes out the patterns and core description files for mode M1 targeting Sub-Block1:

```
// Invoke tool to generate patterns for each mode of top_level
set_context pattern -scan
read_verilog top_level.v // Read netlist of top_level

// Mode M1 consists of the Sub-Block1 design; the user must blackbox Sub-Block2 and apply
// constraints to access Sub-Block1
set_current_mode M1 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM1.ascii
write_core_description top_levelM1.tcd
```

The second dofile reads in the design and writes out the patterns and core description files for mode M2 targeting Sub-Block2:

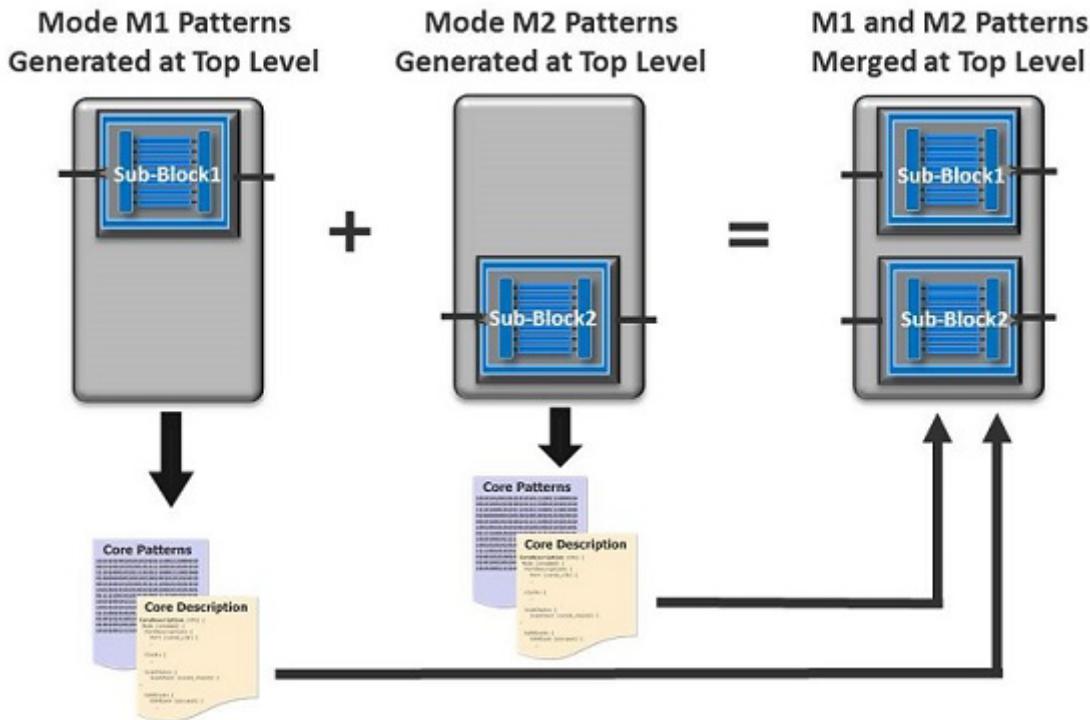
```
// Mode M2 consists of the Sub-Block2 design; the user must blackbox Sub-Block1 and apply
// constraints to access Sub-Block2
set_current_mode M2 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM2.ascii
write_core_description top_levelM2.tcd
```

The third dofile merges the two separately generated top-level pattern sets (modes M1 and M2) into a single pattern set:

```
// Invoke the tool again to retarget patterns to top level
set_context patterns -scan_retargeting
...
read_core_description top_levelM1.tcd
read_core_description top_levelM2.tcd
add_core_instance -core top_level -mode M1 -current_design
add_core_instance -core top_level -mode M2 -current_design
set_system_mode analysis

// Mode extracted from ASCII file
read_patterns top_levelM1.ascii
read_patterns top_levelM2.ascii
write_patterns ...
```

Figure 3-2. Merge Patterns Sets into One Patterns Set



Example 3

The following example demonstrates the use of the `add_core_instances -current_design` switch to merge mode M1 of the current design and mode internal of CoreB. In the example, one pattern set is generated at the top level (mode M1 of the top level) to test Sub-Block1, and a second pattern set is generated at the core level of CoreB. The result, as shown in [Figure 3-3](#), is the integration of the two pattern sets, originally generated for mode M1 of the top level and mode internal of CoreB, into the top level.

The first dofile reads in the top level design and writes out the patterns and core description files for mode M1 targeting Sub-Block1.

```
// Invoke tool to generate patterns for mode M1 of the top level
set_context pattern -scan
read_verilog top_level.v                                // Read netlist of top_level

// Mode 1 consists of the Sub-Block1 design; the user must blackbox any other blocks not
// targeted, and apply constraints to access Sub-Block1
set_current_mode M1 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM1.ascii
write_core_description top_levelM1.tcd
```

The second dofile reads in the CoreB design and writes out the patterns and core description files.

```

set_system_mode setup
delete_design
read_verilog CoreB.v
set_current_mode internal -type internal
...
set_system_mode analysis
create_patterns
write_patterns CoreB.ascii
write_core_description CoreB.tcd

```

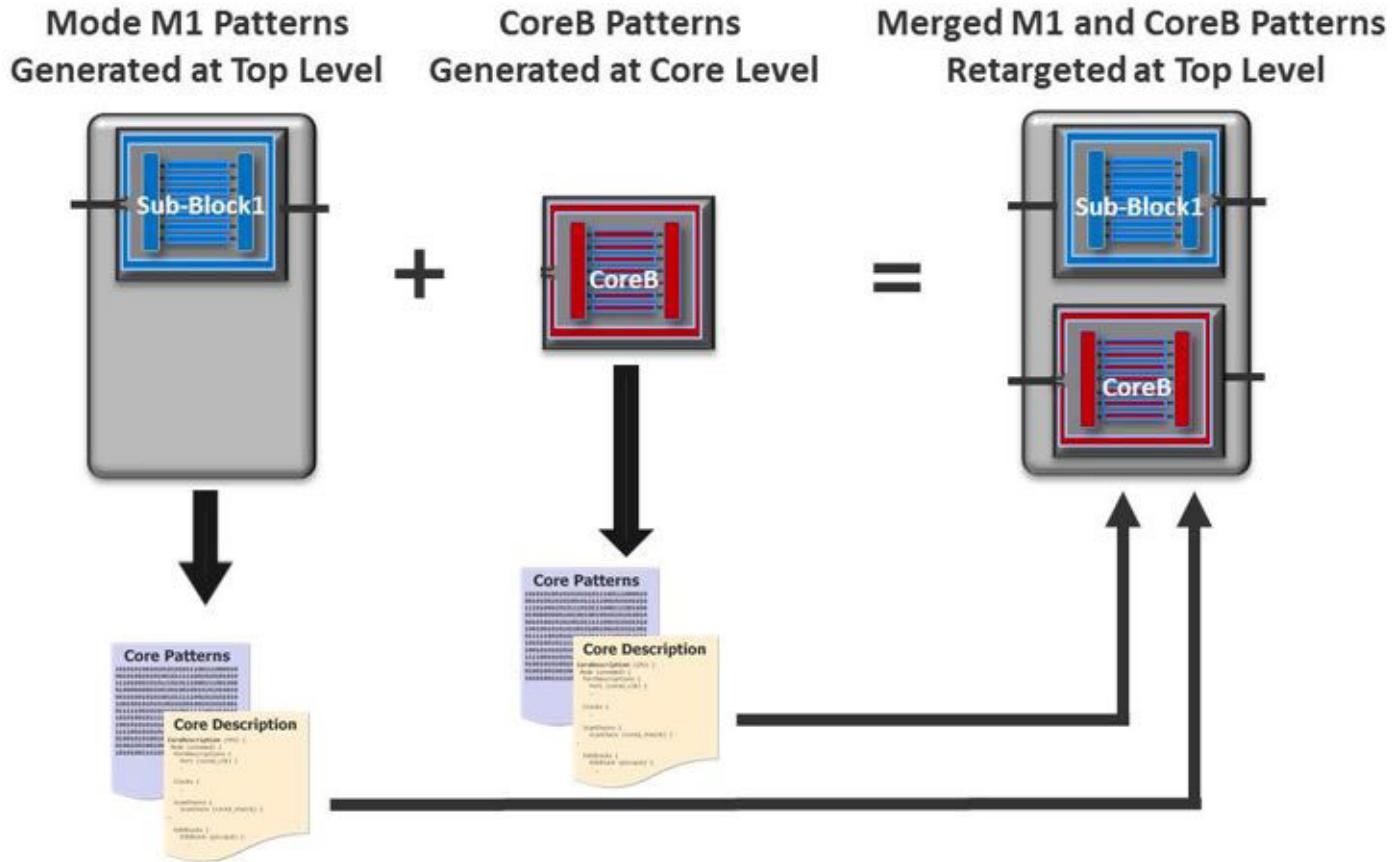
The third dofile merges the top level pattern set (mode M1) with the retargeted CoreB patterns.

```

// Invoke tool to retarget patterns to top level
set_context patterns -scan_retargeting
...
read_core_description top_levelM1.tcd
read_core_description CoreB.tcd
add_core_instance -core top_level -mode M1 -current_design
add_core_instance -core CoreB -mode internal -instances /CoreB_i
...
set_system_mode analysis
read_patterns top_levelM1.ascii
read_patterns CoreB.ascii
write_patterns ....

```

Figure 3-3. Merging Patterns for Top Level and Core at the Top Level



Example 4

The following example shows the four core descriptions in memory that are matched and then used to add core instances:

SETUP> report_core_descriptions

Name	Mode Name	Type
-----	-----	-----
* my_design	retargeting	retargeting
m1231	internal	internal
my_module_name_1	internal	internal
my_module_name_2	internal	internal
my_module_name_3	internal	internal

Note: * denotes current design

The corresponding design modules have the same name as the core descriptions. [Table 3-6](#) shows the mapping between the core descriptions and the design instances:

Table 3-6. Mapping of Core Descriptions and Instances

Core	Instance
m1231	inst_4
my_module_name_1	inst_1
my_module_name_2	inst_2
my_module_name_3	inst_3

The following command adds the core instances:

```
SETUP> add_core_instances -instance inst_*
// Added core instance 'inst_4'.
// Added core instance 'inst_1'.
// Added core instance 'inst_2'.
// Added core instance 'inst_3'.
```

Example 5

The following example shows the four core descriptions in memory that the tool attempts to match and use to add core instances. The matching options, which can be specified and changed with the `set_module_matching_options` command, default to an empty prefix and a “`(_[0-9]+)?`” suffix which means that the tool will try to match the core name against the module name by adding an underscore followed by a digit to it. The name matching mechanism fails to match all of the modules and an error is issued.

SETUP> report_core_descriptions

Name	Mode Name	Type
* my_design	retargeting	retargeting
m123	internal	internal
my_module_name	internal	internal
my_module_name_	internal	internal
my_module_name_2	internal	internal

Note: * denotes current design

The corresponding design modules are:

```
SETUP> get_modules {m123? my_module_name??}
{m1231 my_module_name_1 my_module_name_2 my_module_name_3}
```

Table 3-7 shows the mapping between the core descriptions, modules, and instances:

Table 3-7. Mapping of Core Descriptions With Modules and Instances

Core	Module	Instance
m123	m1231	inst_4
my_module_name	my_module_name_1	inst_1
my_module_name_	my_module_name_3	inst_3
my_module_name_2	my_module_name_2	inst_2

Attempting to add all core instances in one step produces an error as shown below:

```
SETUP> add_core_instances -instance inst_*
// Error: No core description found to match module 'm1231' of instance
// 'inst_4'. Please use -core to specify the core description to use.
```

Example 6

In this example, there are four core descriptions in memory that can be used to add core instances:

SETUP> report_core_descriptions

Name	Mode Name	Type
* my_design	retargeting	retargeting
m1231	internal	internal
my_module_name_1	internal	internal
my_module_name_2	internal	internal
my_module_name_3	internal	internal

Note that * denotes current design

The corresponding design modules have the same name as the core descriptions such that the mapping between the core descriptions and the design instances is as follows:

Core	Instance
m1231	inst_4
my_module_name_1	inst_1
my_module_name_3	inst_3
my_module_name_2	inst_2

The design instance cores are added:

```
SETUP> add_core_instances -instance inst_*
// Added core instance 'inst_4'.
// Added core instance 'inst_1'.
// Added core instance 'inst_2'.
// Added core instance 'inst_3'.
```

Related Topics

[delete_core_instances](#)
[read_core_descriptions](#)
[report_core_descriptions](#)
[report_core_instance_parameters](#)
[report_core_instances](#)
[report_core_parameters](#)
[set_core_instance_parameters](#)
[write_core_description](#)

add_dft_clock_enables

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the `set_current_design` command.

Defines a pin or port of the current design or a port of a sub-module as a clock enable signal that, when activated, disables the clock gating on a clock branch.

Usage

```
add_dft_clock_enables pin_port_net_spec -usage {func_en | test_en} [-active_polarity 0 | 1]
```

Description

Defines a pin, port, or net of the current design or a port of a sub-module as a clock enable signal that, when activated, disables the clock gating on a clock branch.

These clock enable signals are used to suppress clock gating during the functional mode of the operation, and are reused during memory BIST to suppress clock gaters fanning out to memories. On a clock gater cell, the clock enable signal with `-usage func_en` corresponds to the pin typically named “en”; it does not correspond to the pin named “te”, which is used to disable clock gating during the scan shift mode.

Note

 The FE/TE pins of the RTL clock gater are automatically added to the `add_dft_clock_enables` list.

If the clock gater is seen to fanout to the reference of a generated clock, the clock gater is disabled by `all_test` such that the generated clocks gets its reference clock in all test modes. A warning as follows is generated during DRC when this happens:

```
// Note: The clock gated pin 'cgand1/rvlc1_5/Z' fans out to the reference
//        of a generated clock.
//        Disabling it with the DFT signal '~all_test' as its enable is
//        active low.
```

You can use `add_dft_control_points pin -dft_signal_source_name all_test` explicitly on a clock enable pin as long as the use of the `-inverse_dft_control_source` is specified consistently with the active polarity of the clock enable pin.

If you have clock gating cells in your library, you do not need to reference their “en” pins with this command because they are automatically learned. The `report_dft_clock_enables` command reports those automatically if the cell is used at least once in your design. Note that you can also introspect the learned clock gating cells in your design by using the following command:

```
get_modules -filter "cell_type == clock_gating_and || cell_type == clock_gating_or"
```

To see the functional enable port, use:

```
get_ports -of_module $mod -filter "function==func_enable || function==func_enable_inv"
```

To see the test enable port, use:

```
get_ports -of_module $mod -filter "function==test_enable || function==test_enable_inv"
```

You only need to use the `add_dft_clock_enables` command if you have clock gating implemented with logic that does not use clock gating cells such as when you are performing memory BIST insertion in RTL mode. The examples in [Table 3-8](#) illustrate how to define clock enable signals for different scenarios.

The clock enable signals you specify are used during DRC when you issue the `check_design_rules` command in dft context, as long as the “`get_dft_specification_requirements -memory_bist`” command returns “auto” at the time the `check_design_rules` command is issued. Clock enable signals that affect the clocks of memories are automatically identified in DRC and equipped with disabling logic when the `process_dft_specification` command is issued in analysis mode.

[Table 3-8](#) illustrates the logic inserted for a clock enable signal with and without a functional source and with both polarities. The source for these examples is either the nonscan or the `nonscan_inv` port of a special TDR register. A single TDR is used even if you do more than one insertion pass. Clock gaters fanning out to memories tested by memory BIST controllers inserted in a different pass will share the special TDR inserted in the first pass.

Refer to the [Patterns/DftControlSettings](#) property to understand how it is automatically controlled during the execution of the `process_patterns_specification` command.

Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations

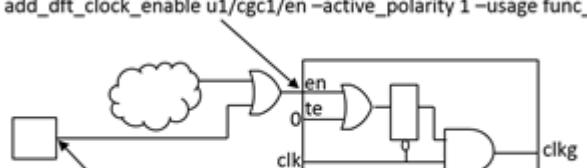
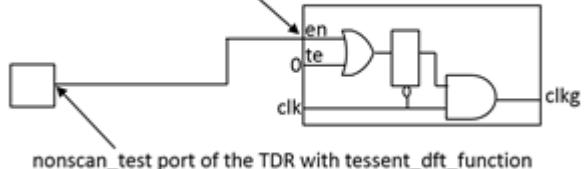
Specification	Resulting DFT
A clock enable with a polarity of 1 and having a functional source prior to the DFT insertion pass	<pre>add_dft_clock_enable u1/cgc1/en -active_polarity 1 -usage func_en</pre>  <p>nonscan_test port of the TDR with <code>tessent_dft_function</code> attribute equal to “<code>scan_tested_instrument_dft_control</code>”</p>
A clock enable with a polarity of 1 and having no functional source (it is tied off or floating) prior to the DFT insertion pass	<pre>add_dft_clock_enable u1/cgc1/en -active_polarity 1 -usage func_en</pre>  <p>nonscan_test port of the TDR with <code>tessent_dft_function</code> attribute equal to “<code>scan_tested_instrument_dft_control</code>”</p>

Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations

Specification	Resulting DFT
A clock enable with a polarity of 0 and having a functional source prior to the DFT insertion pass	<p>add_dft_clock_enable u1/cgc1/en -active_polarity 0 -usage func_en</p> <p>nonscan_test_inv port of the TDR with tessent_dft_function attribute equal to "scan_tested_instrument_dft_control"</p>
A clock enable with a polarity of 0 and having no functional source (it is tied off or floating) prior to the DFT insertion pass	<p>add_dft_clock_enable u1/cgc1/en -active_polarity 0 -usage func_en</p> <p>nonscan_test_inv port of the TDR with tessent_dft_function attribute equal to "scan_tested_instrument_dft_control"</p>

Arguments

- *pin_port_net_spec*

A required string that specifies a list of one or more names of a pin, port, or net object, or a collection containing one or more pin, port, or net objects. If you have an RTL clock gating module, you can specify it on the port of the clock gating module using the commands shown here:

```
set mods [get_modules my_clock_gate*]
add_dft_clock_enables [get_ports en -of_modules $mods] \
    -usage func_en
```

As previously described, the DRC process automatically identifies the instances that need to be disabled during memory BIST and leaves all other instances untouched. The clock gating cells loaded with the [read_cell_library](#) command are automatically learned and do not need to be referenced by this command.

If the object is a port on the current design and the port connects to the pad_io function of a pad cell, the DFT connection will be inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it will be connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it will connect to the from_sji_mux function when it exists, otherwise it will connect to the from_pad function.

- **-usage func_en | test_en**

A required switch and value pair that specifies the usage of the clock enable pins and ports. The func_en value is used for pre-DFT DRC of memory BIST. The test_en value is used for the pre-DFT DRC for logic test.

- **-active_polarity 0 | 1**

An optional switch and value pair that specifies the active polarity of the clock enable signal. When unspecified, the polarity defaults to 1. A polarity of 0 means the clock gating cell is not gating the clock when the clock enable signal is high.

Examples

The following example uses the `add_dft_clock_enables` command to identify the en ports of clock gating modules whose module name starts with “my_clock_gate”, and to define the pin on an instance as a clock enable signal with polarity 0. The source of the clock enable signal is a special TDR that is scan tested with the rest of the functional logic. The [add_dft_control_points](#) command is used to control the select of an existing clock multiplexer; the `add_dft_clock_enables` command is not used for that signal because the clock multiplexer must be controlled in scan test mode too.

The [report_dft_clock_enables](#) command is used in analysis mode with and without the `-post_drc` option to show the specified clock enables and those extracted during DRC. The specified ones can exist on pins or ports of the current design or on ports of any sub-module. During DRC, the clock enables specified on ports of sub-modules are mapped to pins. Only the clock enable signals that affect the memory clocking are kept.

In analysis mode, a read-only wrapper in the tcd partition called “Core(<design_name>)/DftInfo(<design_id>)” contains the ClockEnable signals extracted by DRC. These pins will have DFT added to them to disable the clock gating during memory BIST.

```

set_context dft -rtl
# read design command here

set_current_design my_design
set_design_level physical_block
set_dft_specification_requirements -memory_test on

add_clocks clka -period 6.7ns

set mods [get_modules my_clock_gate*]
add_dft_clock_enables [get_ports en -of_modules $mods] -usage func_en
add_dft_clock_enables clock_ctrl/disable -usage func_en \
    -active_polarity 0
add_dft_control_point clock_mux/select

check_design_rules

report_dft_clock_enables

// Dft clock enable for usage 'func_en'
// -----
//      Node          polarity
// -----
// Port 'en' of 'my_clock_gate_and'      1
// Port 'en' of 'my_clock_gate_or'       1
// Pin  '/clock_ctrl/disable'           0

```

```

report_dft_clock_enables -post_drc

// Dft clock enable for usage 'func_en'
// =====
// -----
//           Node          polarity
// -----
// '/clock_ctl/master_clock_gate/en'      1
// '/uart/mem_clk_gate1/en'              1
// '/clock_ctrl/disable'                 0

report_config_data Core(my_design)/DfTInfo(rtl)/ClockEnables -partition tcd

ClockEnables {
    ClockEnable(/clock_ctl/master_clock_gate/en) {
        active_polarity : 1;
    }
    ClockEnable(/uart/mem_clk_gate1/en) {
        active_polarity : 1;
    }
    ClockEnable(/clock_ctrl/disable) {
        active_polarity : 0;
    }
}

```

Related Topics

[add_dft_clock_mux](#)
[add_dft_control_points](#)
[read_cell_library](#)
[delete_dft_clock_enables](#)
[report_dft_clock_enables](#)

add_dft_clock_mux

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set.

Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the [process_dft_specification](#) command.

Usage

```
add_dft_clock_mux pin_port_net_spec -test_clock_source test_clock_source_spec
    [-dft_signal_source_name dft_signal_source_name] [-inverse_dft_signal_source]
    [-auto_uniquify]
```

Description

Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the [process_dft_specification](#) command.

The specified pin, port, or net is intercepted with the 0 input of the multiplexer and the 1 input is connected to the specified *test_clock_source_spec*. The select of the multiplexer is connected to the specified *dft_signal_source_name* with an optional inversion based on the presence of the *-inverse_dft_signal_source* switch.

The default *dft_signal_source_name* value is “all_test” which is a DFT signal that is automatically asserted high during all test modes: those modes based on scan and those that are not. You can also define your own custom *dft_signal_source_names* by using the [register_static_dft_signal_names](#) command.

If you need to insert a custom multiplexer where the select is controlled by an arbitrary source, use the [intercept_connection](#) command with the *-dft_cell_function* option specified as “mux”. Those, however, need to be inserted prior to running DRC.

If you are familiar with the LogicVision Memory Bist flow, a clock multiplexer was specified in ETChecker using the *lv.InternalClockSource -test_clock_source* option. The select was always connected to the *ETClockEnable* signal. In the Tessent shell flow, the name of the *ETClockEnable* signal is “all_test” because it is a test signal that is activated during all test modes, whether they are based on scan or not.

You no longer get an error if you issue the exact same *add_dft_clock_mux* twice. You only get an error if the second specification has one or more options that are different from the first specification.

Arguments

- ***pin_port_net_spec***

A required string argument that specifies the name of a pin, port, or net of the current design or a collection containing one port, pin, or net object of the current design. The insertion of the multiplexer is allowed on both input and output ports and input and output pins. The select is asserted to the `default_value_in_all_test` value of the `dft_signal_source_name` during pre-DFT DRC, which is 1 when using `all_test` as the DFT signal source name. All combinations of clock path are verified during `extract_icl`. The `default_value_in_all_test` value of the `dft_signal_source_name` is specified with the [register_static_dft_signal_names](#) command.

- **-test_clock_source *test_clock_source_spec***

A required switch-value pair that is used to specify the name of a port, pin, or net of the current design or a collection containing one port, pin, or net object. The specified `test_clock_source_spec` is used to source the 1 input of the multiplexer. The 0 input of the multiplexer remains the original source of the `pin_or_port_spec`.

If the object is a port on the current design and the port connects to the `pad_io` function of a pad cell, the DFT connection will be inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it will be connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it will connect to the `from_sji_mux` function when it exists otherwise it will connect to the `from_pad` function.

- **-dft_signal_source_name *dft_signal_source_name***

An optional switch-value pair that is used to specify a string that defines the name of a DFT signal that sources the select for the multiplexer.

The default `dft_signal_source_name` value is “`all_test`” which is a DFT signal that is automatically asserted high during all test modes: those modes based on scan and those that are not. You can also define your own custom `dft_signal_source_name` using the [register_static_dft_signal_names](#) command.

- **-inverse_dft_signal_source**

An optional Boolean option that specifies an inversion between the source DFT signal and the select port of the multiplexer. The clock multiplexer always needs to reset selecting the 0 input of the multiplexer otherwise the insertion of the multiplexer would affect the functional mode and fail the pre-DFT versus post-DFT formal verification. To satisfy the requirement, you can only refer to a `dft_signal_source_name` that resets to 0 when the `-inverse_dft_signal_source` option is not specified and you can only refer to a `dft_signal_source_name` that resets to 1 when the `-inverse_dft_signal_source` is specified.

- **-auto_uniquify**

An optional Boolean option that specifies the parent module of the `pin_port_net_spec` is to be auto-uniquified if seen to be inside an instance of a repeated module. If you do not specify this option, you will get an error when running the [check_design_rules](#) command and you did not specify an `add_dft_clock_mux` command for all instances of the repeated

module. The *test_clock_source_spec* and the *dft_signal_source_name* are allowed to be different for each instance. The error message lists the missing instances making it easy to add them and avoid unification.

Examples

The following example shows the use of three `add_dft_clock_mux` commands. One is to enable the sharing of a common clock source during memory BIST, a technique often used to minimize the number of clock sources needed during manufacturing test. The second one is to enable bypassing of the PLL based on a user-defined DFT signal source name called `pll_bypass`. The third one is used to provide an alternative source for the reference of the PLL based on a user-defined DFT signal source name called `alt_ref`. Notice that during insertion the interception for all multiplexers is done first and then the connections for the test clock sources are performed. This makes sure the test clock source of the green multiplexer is not intercepted by the red interception. Notice that it is not allowed to insert a clock multiplexer in the middle of a balanced clock tree. This is why this example has two declared branch clocks to denote the base of the balanced clock tree feeding the memories. The inserted clock multiplexers can only fanout to the reference of generated clocks and to branch clocks but not to memories or scannable flip-flops directly. If you want to insert a multiplexer without declaring a branch clock, you have to insert it in your design (either manually or with a custom insertion script using the `intercept_connection` command). You leave the select tied off and then point to it using the `add_dft_control_points` to make it controllable by IJTAG.

Notice how the `DftSpecification` contains a `Tdr` wrapper having an Attribute “`tesson_dft_function`” set to “`scan_resource_instrument_dft_control`”. You cannot delete or modify this `Tdr` wrapper when you use the `add_dft_clock_mux` and/or the `add_dft_control_points` commands.

```

> set_context dft -rtl
> #read design command here
> set_current_design my_design
> set_design_level physical_block
> set_dft_specification_requirements -memory_test on
> add_clocks clka -period 6.7ns
> add_clocks vco_buf/Y -branch
> add_clocks clkbuf/Y -branch
> register_static_dft_signal_names alt_ref pll_bypass
> add_dft_clock_mux clkbuf -test_clock_source mem1/clk \
    -dft_signal_source_name all_test
> add_dft_clock_mux pll1/vco -test_clock_source pll1/ref \
    -dft_signal_source_name pll_bypass
> add_dft_clock_mux pll1/ref -test_clock_source clkbuf \
    -dft_signal_source_name alt_ref
> report_dft_clock_mux
// Dft clock muxes/ =====
// -----
// Node Test      clock source      Control source name
// -----
// 'clkbuf'       mem1/clk        all_test
// 'pll1/vco'     pll1/ref        pll_bypass
// 'pll1/ref'     clkbuf         alt_ref

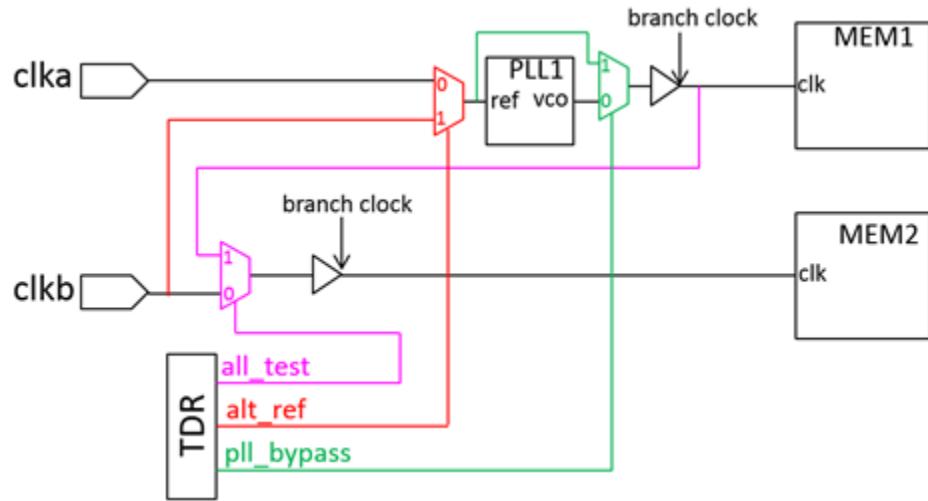
> check_design_rules
> set spec [create_dft_specification]
> report_config_data $spec

DftSpecification(mydesign,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Sib(sri) {
                Attributes {
                    tessent_dft_function : scan_resource_instrument_host;
                }
                Tdr(sri_ctrl) {
                    Attributes {
                        tessent_dft_function : scan_resource_instrument_dft_control;
                    }
                }
            }
        }
    }
}

> process_dft_specification

```

Figure 3-4. Example Usage of add_dft_clock_mux



Related Topics

[add_dft_clock_enables](#)
[report_dft_clock_muxes](#)
[delete_dft_clock_muxes](#)

[add_dft_control_points](#)

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set.

Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the [process_dft_specification](#) command.

Usage

```
add_dft_control_points pin_port_net_spec [-type type] [-inverse_dft_signal_source]  
[-dft_signal_source_name dft_signal_source_name] [-ignore_existing_sources]  
[-auto_uniquify] [-allow_editing_below_ijtag_instances]
```

Description

Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the [process_dft_specification](#) command.

For DFT control points of type “static_dft_control”, when the specified object is a port or an output pin, and when the specified object is an input pin or a net with a functional source (that is, its `has_functional_source` [Pin](#) or [Net](#) attribute is true), it is intercepted with the 0 input of a multiplexer and the 1 input gets connected to the static DFT signal source location. The source is inverted when the `-inverse_dft_signal_source` option is specified. The select of the multiplexer is connected to the “all_test” DFT signal. The logic is a simple OR gate or AND gate when the `dft_signal_source_name` is specified as “all_test”. When the specified pin or port has no functional source (that is, its `has_functional_source` attribute is false), it gets directly connected to the DFT signal source with an optional inversion. See [Table 3-9](#) for illustrations of the inserted logic and connections. If the specified node is tied but the value does not match the reset value of the DFT signal, you will get an error unless you use the `-ignore_existing_sources`.

The default `dft_signal_source_name` value is “all_test”, which is a DFT signal that is automatically asserted high during all test modes: modes based on scan and those that are not. Use [report_dft_signal_names](#) to see the built-in static DFT signals. You can also define your own custom DFT signal names using the [register_static_dft_signal_names](#) command. For more information about the custom `dft_signal_source_names` and their usage, see the [register_static_dft_signal_names](#) command section.

If you need to insert a custom gate where the second data input and the select input is controlled by arbitrary sources, use the [intercept_connection](#) command with the `-dft_cell_function` option specified as “mux”, “and” or “or”. Those, however, needs to be inserted prior to running DRC.

If you are familiar with the LogicVision MemoryBIST flow, a control point was specified in ETChecker using the `lv.ETClockEnable` and the `lv.InjectControl -type UserBit` properties. In the Tesson Shell flow, the ETClockEnable signal is now called “all_test” because it is a test

signal that is activated during all test modes, whether they are based on scan or not. A UserBit is now a user-registered DFT signal name.

For DFT control point of type “dynamic_dft_control”, the specified signal is always directly connected to the specified node. You will get an error if the specified node has a functional source (that is, its has_functional_source attribute is true) or if it is tied to a 1 unless you use the -ignore_existing_sources switch. In such a case, the existing source is simply replaced by the dynamic DFT signal source.

You no longer get an error if you issue the exact same add_dft_control_point twice. You only get an error if the second specification has one or more options that are different from the first specification.

In a hierarchical flow, if you add any dynamic DFT signals in the current design and the current design instantiates some lower designs in which you had added dynamic DFT signals sourced by ports, an “add_dft_control_point -type dynamic_dft_signal” command will be inferred to connect them if there "has_functional_source" attribute returns true. When this happens, you will see a note similar to the following:

check_design_rules

```
// Note: Inferred 5 'add_dft_control_points <pin>
//        -type dynamic_dft_control' commands to connect the dynamic DFT
//        signals pins on the child block instances. Use
//        report_dft_control_points to see them.
```

If the scan_en dynamic DFT signal was not added yet and the scan_en DFT signals are seen to be floating at the input of a sub designs, they will be tied low as part of running the [process_dft_specification](#) command such that the functional and the non-scan test modes work. The scan_en will later be connected to the DFT signal “scan_en” once you add it in the current design.

Table 3-9. Dft Logic for add_dft_control_points Nodes

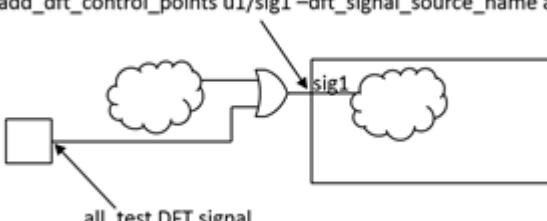
Specification	Resulting DFT
A dft control point with DFT signal source equal to “all_test”. The object has a functional source prior to the DFT insertion.	<p>add_dft_control_points u1/sig1 -dft_signal_source_name all_test</p>  <p>The diagram illustrates the resulting DFT logic. A square symbol representing a functional source is connected to a cloud-shaped terminal labeled "sig1". This connection passes through a switch, indicated by a T-junction with two paths. One path leads to the switch, and the other path originates from the switch and connects to the "sig1" terminal. Below the switch, an arrow points to the text "all_test DFT signal", indicating that the switch is controlled by this signal.</p>

Table 3-9. Dft Logic for add_dft_control_points Nodes (cont.)

Specification	Resulting DFT
A dft control point with control source equal to a name other than “all_test”. The object has a functional source prior to the DFT insertion.	<pre>add_dft_control_points u1/sig1 -dft_signal_source_name <name></pre> <p>all_test and <name> DFT signals</p>
A dft control point with an inverted control source equal to “all_test”. The object has a functional source prior to the DFT insertion.	<pre>add_dft_control_points u1/sig1 -dft_signal_source_name all_test -inverse_dft_signal_source</pre> <p>all_test DFT signal</p>
A dft control point with an inverted control source equal to a name other than “all_test”. The object has a functional source prior to the DFT insertion.	<pre>add_dft_control_points u1/sig1 -dft_signal_source_name <name> -inverse_dft_signal_source</pre> <p>all_test and <name> DFT signals</p>
A dft control point with any DFT signal source name. The object has no functional source (it is tied or floating) prior to the DFT insertion.	<pre>add_dft_control_points u1/sig1 -dft_signal_source_name <name></pre> <p><name> DFT signal</p>
A dft control point with any inverted DFT signal source name. The object has no functional source (it is tied or floating) prior to the DFT insertion.	<pre>add_dft_control_points u1/sig1 -dft_signal_source_name <name> -inverse_dft_signal_source</pre> <p><name> DFT signal</p>

Arguments

- ***pin_port_net_spec***

A required string that specifies a list of one or more names of pin, port, or net object, or a collection containing one or more pin, port, or net objects. The control point is allowed to be specified on both input and output ports and input and output pins. The DFT signal source name “all_test” is asserted active during pre-DFT Memory BIST DRC. The other DFT signal source names are asserted to their respective “default_value_in_all_test” specified when registering them. See the [register_static_dft_signal_names](#) command for more information about the behavior of the DFT signals.

If the object is a port on the current design and the port connects to the pad_io function of a pad cell, the DFT connection will be inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it will be connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it will connect to the from_sji_mux function when it exists otherwise it will connect to the from_pad function.

- **-type static_dft_control | async_set_reset | dynamic_dft_control**

An optional switch-value pair that specifies the kind of DFT control to add. For DFT control point of type static_dft_control, the specified node is connected or intercepted with a multiplexer by the specified DFT signal as shown in [Table 3-9](#). For DFT control of type dynamic_dft_control, the specified node is connected to the specified DFT signal.

When type is set to async_set_reset, the specified node will be checked by [DFT_C9](#) and if seen to source asynchronous set or reset pins of one or more scannable flip-flops, the common off value will be determined, and an OR gate or an AND gate will be inserted to allow disabling the asynchronous set and reset pins during shift mode. An OR gate is used when the active polarity is determined to be 0 and an AND gate is used when the active polarity is determined to be 1. The other input of the GATE is connected to the DFT signal “async_set_reset_dynamic_disable”. See [Table 3-11](#) on page 209 in the [add_dft_signals](#) command description for more information about this DFT signal. By default, DFT_C9 will infer the async_set_reset DFT control points on the nodes found to be controlling sources of asynchronous set or reset pins. See “[set_drc_handling dft_c9 -auto_fix](#)” command description to know how to disable the auto fixes.

If you have issued the `add_dft_control_point -type async_set_reset` command and `set_dft_specification_requirements -logic_test` is off, then issuing the `check_design_rules` command will produce the following error:

```
// Error: You can't have 'add_dft_signals -type async_set_reset'
// when 'set_dft_specification_requirements -logic_test' is 'off'.
// Unless DFT_C9 runs, the active polarity of the set or reset
// nodes cannot be determined.
// Either add '-logic_test on' to your
// set_dft_specification_requirements settings or issue
// delete_dft_control_points {{rst} {ram1/AR[1]}}
```

- **-inverse_dft_signal_source**

An optional Boolean option that specifies an inversion between the DFT signal specified by the `-dft_signal_source_name` option and the ***pin_port_net_spec***.

- **-dft_signal_source_name *dft_signal_source_name***

An optional switch-value pair that is used to specify a string that defines the name of DFT signal that is to source the ***pin_port_net_spec*** during test modes.

The default *dft_signal_source_name* value is “all_test”, which is a DFT signal that is automatically asserted high during all test modes: modes based on scan and those that are not. You can also define your own custom *dft_signal_source_names* using the [register_static_dft_signal_names](#) command.

- **-ignore_existing_sources**

An optional Boolean option that is used to suppress the error that is normally generated in those two situations. For DFT signal of type “static_dft_control”, you will get an error if the specified node is tied and the tie value is not consistent with the reset value of the DFT signal taking into account if the source is inverted or not. Specifying this switch suppresses the error and allows replacing the tie value with the connection from the DFT signal.

Realize that suppressing this error will cause the node to have a different value in functional mode as compared to before the connection was made.

For DFT signal of type “dynamic_dft_control”, you will get an error if the specified node is tied and the tie value is not consistent with the reset value of the DFT signal taking into account if the source is inverted or not. You will also get an error if the node already has a functional source. Specifying this switch suppresses the error and allows replacing the tie value or the functional source with the connection from the DFT signal. Realize that suppressing this error will cause the node to have a different value in functional mode as compared to before the connection was made.

- **-auto_uniquify**

An optional Boolean option that specifies the parent module of the ***pin_port_net_spec*** is to be auto-uniquified if seen to be inside an instance of a repeated module. If you do not specify this option, you will get an error when running the [check_design_rules](#) command and you did not specify an `add_dft_control_points` command for all instances of the repeated module. The *dft_signal_source_name* is allowed to be different for each instance. The error message lists the missing instances making it easy to add them and avoid unification.

- **-allow_editing_below_ijtag_instances**

An optional Boolean switch that allows you to point to a pin that is below a module that has a matched ICL module. By default, you are not allowed to point to a pin that is inside an instance with its `is_hard_module` or `created_by_synthesis` attribute set to true. When `-type` is set to `static_dft_control`, you are also not allowed to point to a pin that is below a module that has a matched ICL module because those DFT inject points can affect the ICL view of the module. If you are sure that the control point will not affect the ICL view, you can use

this switch to relax the check. Otherwise, go back to a step prior to when the ICL was created to insert the DFT control point.

Examples

This example uses the `add_dft_control_points` to inject a control multiplexer on the select of an existing clock multiplexer. The inserted logic is illustrated in [Figure 3-5](#). The output port “`all_test`” on the TDR is automatically asserted high at the beginning of the patterns created using [process_patterns_specification](#). You can decide to activate the PLL bypass mode by setting the `pll_bypass` port active using the [Patterns/DftControlSettings](#) wrapper. If you have a programmable PLL with a built-in `bypass_en` port and ports to control the multiplication factor, you can use this command to control those ports. A more automated way is to create an ICL model for the PLL describing the `DataInPorts` of the PLL. The [create_dft_specification](#) command will automatically hook them up to the `ijtag` network. You can then use a [ProcedureStep](#) to program your PLL. You must use the `add_dft_control_points` when the injected values are needed by the pre-DFT DRC.

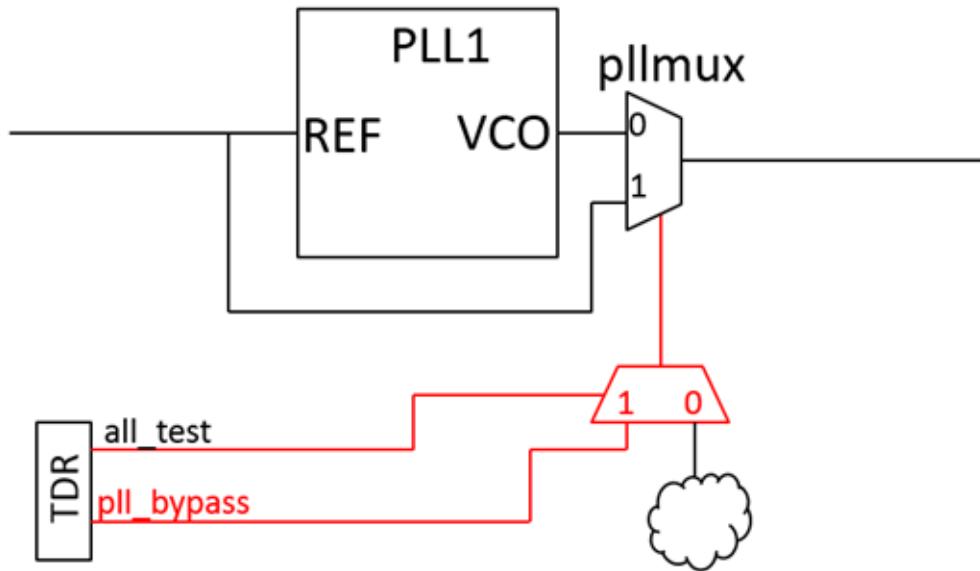
Notice how the `DftSpecification` contains a `Tdr` wrapper having an Attribute “`tessent_dft_function`” set to “`scan_resource_instrument_dft_control`”. You cannot delete or modify this `Tdr` wrapper when you use the `add_dft_clock_mux` and/or the `add_dft_control_points` commands.

```
set_context dft -rtl  
  
#read design command here  
  
set_current_design my_design  
set_design_level physical_block  
set_dft_specification_requirements -memory_test on  
add_clocks clka -period 6.7ns  
add_clocks pllmux/y -reference PLL1/REF -freq_multiplier 10  
register_static_dft_signal_names pll_bypass  
add_dft_control_points pllmux/s -dft_signal_source_name pll_bypass  
  
check_design_rules  
  
set spec [create_dft_specification]  
report_config_data $spec
```

```
DftSpecification(my_design,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Sib(sri) {
                Attributes {
                    tessent_dft_function : scan_resource_instrument_host;
                }
                Tdr(sri_ctrl) {
                    Attributes {
                        tessent_dft_function : scan_resource_instrument_dft_control;
                    }
                }
            }
        }
    }
}

process_dft_specification
```

Figure 3-5. Example Usage of add_dft_control_points Command



Related Topics

[add_dft_clock_enables](#)
[add_dft_clock_mux](#)
[process_dft_specification](#)
[delete_dft_control_points](#)
[report_dft_control_points](#)

[add_dft_modal_connections](#)

Context: dft

Mode: setup

A command used to insert modal connections from specified output data source nodes or to input data destination nodes typically used to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports.

Usage

```
add_dft_modal_connections  
  {-ports port_spec | -auxiliary_data_pins pin_spec [-auxiliary_enable_pins pin_spec] }  
  {-output_data_source_nodes port_pin_spec  
   | -input_data_destination_nodes port_pin_spec}  
  {-enable_dft_signal dft_signal_name | -enable_source_node port_pin_spec}  
  [-inverse_enable]  
  [-pipeline_stages pipeline_stages | -nonscan]
```

Description

A command used to insert modal connections from specified output data source nodes or to input data destination nodes. Once all modal connections are added, they are analyzed and appropriate multiplexing and pipelining logic is inserted to implement the specification. You can use this command to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports. You use the [report_dft_modal_connections](#) command to see the cumulative effect of your specified add_dft_modal_connections commands and understand the multiplexing logic that will be inserted.

You typically have a limited amount of top-level ports that you can use to provide data to the EDT channel inputs and observe the EDT channel outputs. This typically prevents you from running all EDT cores in parallel. Instead you need to use sets of cores to create multiple top level access modes so that you do not exceed the TOP level resource available in your chip.

The add_dft_modal_connections command is very flexible, and allows you to automatically insert the necessary multiplexing and pipelining logic. As shown in [Figure 3-6](#), when you specify multiple modal connections on the same output port you will need multiplexing logic for that port. The blue logic represents the auxiliary output logic present on the output port. The auxiliary output logic must allow multiplexing between the auxiliary data out values and the functional values, and allow the pad output buffers to be enabled. You can use the [BoundaryScan](#) wrapper of the [DftSpecification](#) to equip the top-level ports with auxiliary input and output logic. The actual logic inserted is a bit more complicated than the blue logic shown in [Figure 3-6](#) because Boundary Scan also needs control of the ports, and you do not want to cascade two multiplexers along the functional data path. See [Figure 10-29](#) on page 3368 in the [AuxiliaryInputOutputPorts](#) section of the [BoundaryScan](#) wrapper section for precise description of this logic.

Note

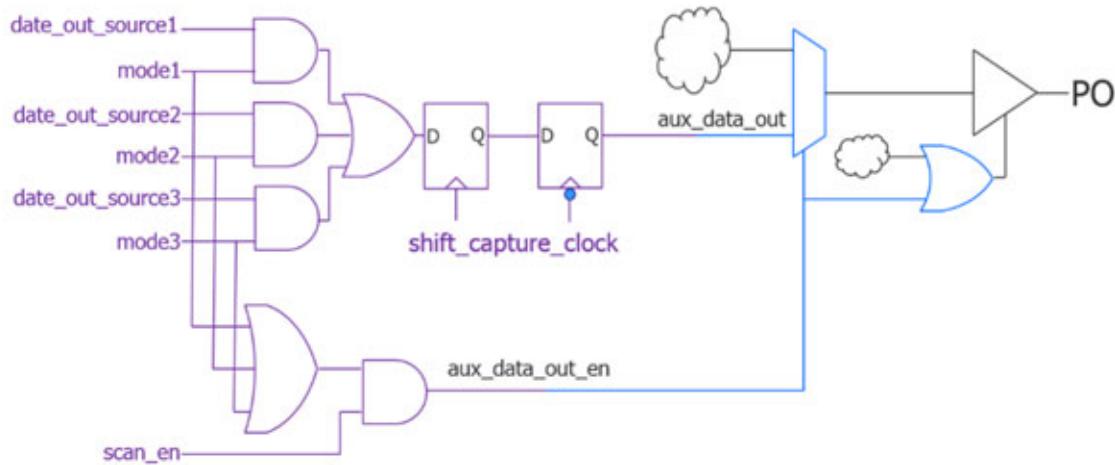
 The circuitry described next is included in modules handling one bit at a time. Those modules are instantiated in the parent instance of the auxiliary data pin they handle. If you used the [BoundaryScan AuxiliaryInputOutputPorts](#) wrapper to insert your auxiliary data logic, the auxiliary data logic will be located in the logical grouping module associated to a given port. You have the flexibility to place the logical grouping modules where you want. The modal connection modules will follow the logical grouping module and be instantiated in the same parent instance module. If you use the `-auxiliary_data_pins` option, the modules are instantiated in the parent instance of those pins. For the modules sometime created at the destination of auxiliary input connections and illustrated in [Figure 3-8](#), the parent instance of the “`input_data_destination_nodes`” is where they are instantiated.

Note

 The modal connection modules are created to handle one bit at a time. The same module will be instantiated multiple times if there are multiple connections having the exact same muxing and pipelining requirement unless the [reuse_modules_when_possible](#) property found in the [DftSpecification](#) wrapper defaults or is set to off in which case one module is created per bit even if the content is identical to others.

The multiplexing is achieved by ANDing each data source with its associated enable source and ORing the AND gates together. This kind of multiplexing is very efficient for routing as there is no select priority logic, and the layout tool is capable of splitting the OR gate to minimize the congestion. The enable sources are also ORed together to activate the auxiliary output data path. Notice that the auxiliary data output enable signal is gated by `scan_en`. This allows the functional output logic to be observable during the capture cycles either by the boundary scan cell if you are using the boundary scan during scan test (see the [max_segment_length_for_logictest](#) property in the [BoundaryScan](#) wrapper to enable that) or by the PO itself.

Figure 3-6. Multiplexing and Pipelining Logic for Output Connections

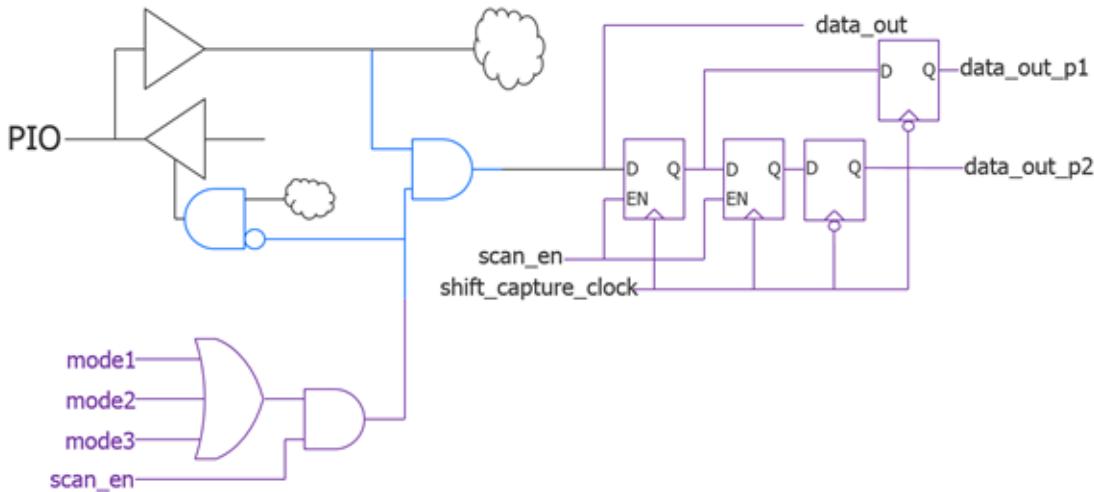


(Output data muxing and pipelining logic added by add_dft_modal_connections)
(Auxiliary Data muxing such as the one added by Tesson Boundary Scan)

Multiplexing logic is not needed at the source for the auxiliary input data connections. As shown in [Figure 3-7](#), the enable sources still need to be ORed together to drive the auxiliary input enable pin of the port. The auxiliary input enable is used to gate off the auxiliary data so that it does not follow the port value during functional mode. When the port is an inout port, the enable is also used to turn off the output driver so that the port can be used to supply input data. Finally, if pipelining was requested, the auxiliary data is pipelined and the re-timed pipelined sources are used to make the connections to the specified destinations.

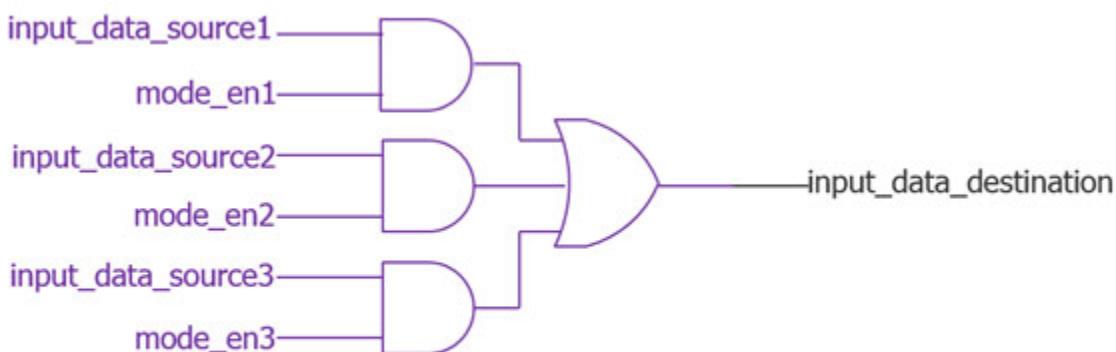
As shown in [Figure 3-8](#), it is possible that multiplexing logic is required at the destination nodes. For example, if an ELT core is re-used in multiple configurations, the edt_channel_in ports may have different sources in different modes. The circuit shown in [Figure 3-8](#) is used to implement the necessary multiplexing. When such multiplexing is required, the [report_dft_modal_connections](#) command will include an “Input destinations” section showing the multiplexing on the destination pins.

Figure 3-7. Enable Logic and Pipelining for Input Connections



(Input data enabling and pipelining logic added by add_dft_modal_connections)
(Auxiliary Data enabling such as the one added by Tesson Boundary Scan)

Figure 3-8. Multiplexing for Destinations of Modal Connections



(Input data muxing added by add_dft_modal_connections)

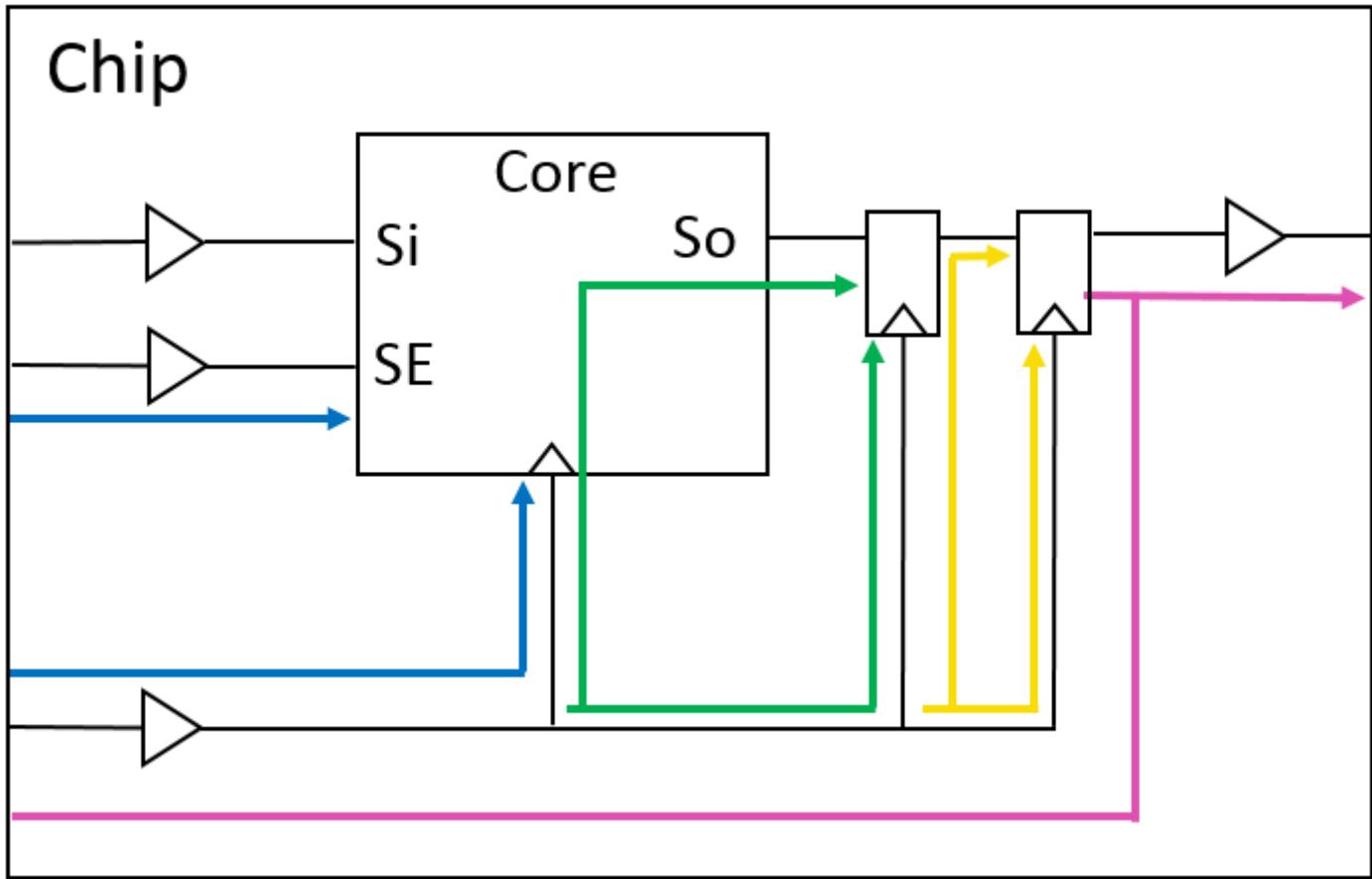
Notice that the pipelining flip-flops are clocked using the shift_capture_clock which you define the source using the “[add_dft_signals shift_capture_clock](#)” command. If you are using the add_dft_modal_connections command to provide source or observation of data coming from non-scan test modes, you specify the -nonscan switch and you do not specify the -pipeline_stages switch.

Maximizing the shift rate of the logictest mode

The shift clock rate of the scan-based test modes is often limited by the loop timing arc (illustrated by the magenta arrow in [Figure 3-9](#)) starting from the shift clock input, propagating through the clock tree to reach the last flop on the chain and exiting the chip through an output buffer. The majority of the delay is spent in the clock tree. To reduce this timing arc, you can use a few pipelining flip-flops in series for the scan-out path and control the clock tree synthesis

tool to place the last flop as close as possible to the base of the clock tree. If the clock tree delay is larger than one clock period, use a clock point that is mid-point along the clock tree for the middle pipelining flop. You then have a full cycle minus the clock delay difference between the first and the middle pipelining flip-flop (illustrated by the green arcs), and again a full cycle minus the clock delay difference between the middle and the last pipelining flip-flop (illustrated by the yellow arcs). The `add_dft_modal_connections` command allows you to insert up to five pipelining flops in series. You then need to provide instruction to your Clock tree synthesis to not balance the clock inputs to the last stages in order to allow stepping down toward the output port. Pipelining is typically not needed on the input direction because the clock path delay largely compensates for the delay in the data paths (illustrated by the blue arcs).

Figure 3-9. Timing Arcs for Scan In and Out Paths



Arguments

- **-ports *port_spec***

A switch used to specify a list of one or more port names or a collection of one or more port objects that are to be used as data source or destination. The direction of the data connection is controlled by the use of one of the mandatory and mutually exclusive `-output_data_source_nodes` and `-input_data_destination_nodes` switches. When the `-output_data_source_nodes` switch is used, the specified ports must either be existing or to-be-created output ports or existing output or inout ports equipped with auxiliary data

logic. When design level, as specified by the [set_design_level](#) command is “chip”, ports cannot be created and must already be connected to a pad equipped with auxiliary data output logic. At the physical or sub-block level, you may specify to-be-created ports or existing ports having no existing functional sources. See the [AuxiliaryInputOutputPorts](#) section of the [BoundaryScan](#) or [EmbeddedBoundaryScan](#) sections to learn how to automatically equip ports with auxiliary input or output data logic.

When you use Tessent Boundary Scan to insert the auxiliary input and output data logic, the location of the auxiliary data and enable pins is described in the [Instrument Dictionary](#) created during boundary scan insertion. You can then use the -ports switch to point to the ports and the mapping to the auxiliary data and enable pins is done automatically. If you implemented the auxiliary data logic manually or with a third party tool, you have two options. You can either describe the mapping between the top level ports and the auxiliary data pins using a Tcl dictionary called “::auxiliary_data_dict” or by pointing explicitly to the auxiliary data and enable pins using the -auxiliary_data_pins/-auxiliary_enable_pins switches for each invocation of the add_dft_modal_connections commands.

The format of the “::auxiliary_data_dict” dictionary is the following:

```
<port_name> {
    auxiliary_output_pin      <pin_name>
    auxiliary_output_enable_pin <pin_name>
    auxiliary_input_pin       <pin_name>
    auxiliary_input_enable_pin <pin_name>
}
... #repeat for all ports
```

You can use the Tcl “dict” command to populate this dictionary once so that you can then make your DFT modal connection using the -ports switch instead of having to explicitly point to the auxiliary data and enable pins explicitly each time. Ex:

```
dict set ::auxiliary_data_dict gpio[1] auxiliary_input_pin u1/aux_in
```

The enable pin entries in the dictionary are optional. If you do not list them, the add_dft_modal_connections command will only perform the data multiplexing and will assume you have already handled the enabling of the auxiliary data path.

- **-auxiliary_data_pins *pin_spec***

A switch used to specify the location of the auxiliary input or output data pins. The *pin_spec* value is a list of one or more names of pin objects or a collection containing one or more pin objects. It refers to auxiliary input data pins when the -input_data_destination_nodes switch is used. It refers to auxiliary output data pins when the -output_data_source_nodes switch is used. If you have created the auxiliary data logic with Tessent Boundary Scan using the [AuxiliaryInputOutputPorts](#) wrapper of the [BoundaryScan](#) or the [EmbeddedBoundaryScan](#) wrappers, or if you have described your auxiliary data and enable pins in the Tcl dictionary “::auxiliary_data_dict” described above in the -ports switch section, you do not need to use the -auxiliary_data_pins switch. Instead., you use the -ports switch and the mapping to the auxiliary data and enable pins will be extracted automatically.

If you have already connected the auxiliary enable pins and you only want the `add_dft_modal_connections` command to handle the data muxing, point to the auxiliary data pin using the `-auxiliary_data_pins` switch and omit specifying the `-auxiliary_enable_pins` switch.

- **`-auxiliary_enable_pins` *pin_spec***

A switch used to specify the location of the auxiliary input or output enable pins. The *pin_spec* value is a list of one or more names of pin objects or a collection containing one or more pin objects. It refers to auxiliary input enable pins when the `-input_data_destination_nodes` switch is used. It refers to auxiliary output enable pins when the `-output_data_source_nodes` switch is used. See the description of the `-auxiliary_data_pins` switch above to know when you need to use it and how the location of the auxiliary enable pins are inferred automatically when using the `-ports` switch.

- **`-output_data_source_nodes` *port_pin_spec***

A switch that is mutually exclusive with the `-input_data_destination_nodes` switch but required when the `-input_data_destination_nodes` switch is not specified. It is used to specify a list of one or more names of port or pin objects or to specify a collection containing one or more port or pin objects that sources the modal connection. The number of pins or ports specified in the *port_pin_spec* must match the number of elements specified with the `-ports` or `-auxiliary_data_pins` switches. If the port or the auxiliary output pin is already connected to the source needed in the given mode, use the symbol “*” as the *port_pin_spec* value and the current source will be preserved for that mode. See [Example 1](#) for an example of its usage.

- **`-input_data_destination_nodes` *port_pin_spec***

A switch that is mutually exclusive with the `-output_data_source_nodes` switch but required when the `-output_data_source_nodes` switch is not specified. It is used to specify a list of one or more names of port or pin objects or to specify a collection containing one or more port or pin objects that are the destination of the modal connections. The number of pins or ports specified in the *port_pin_spec* must match the number of elements specified with the `-ports` or `-auxiliary_data_pins` switches. If the port or the auxiliary input pin is already connected to the needed destination in the given mode, use the symbol “*” as the *port_pin_spec* value and the current destination will be preserved for that mode even if pipelining is inserted. See [Example 1](#) for an example of its usage.

- **`-enable_dft_signal` *dft_signal_name***

A switch that is mutually exclusive with the `-enable_source_node` switch but required when the `-enable_source_node` switch is not specified. It specifies the name of a DFT signal that is to serve as the source to enable the modal connection. When the `-nonscan` switch is not specified, the DFT signal must be of type “`scan_mode`”. When the `-noscan` switch is used, the DFT signal must be of type “`global_dft_control`”. See the [add_dft_signals](#) command for more information about DFT signal types. If the specified *dft_signal_name* is not already present or added, it will be added automatically if it is a registered DFT signal name. Use [report_dft_signals](#) to see the list of registered DFT signal names. See the [register_static_dft_signal_names](#) command description to register your own.

- **-enable_source_node *port_pin_spec***

A switch that is mutually exclusive with the -enable_dft_signal switch but required when the -enable_dft_signal switch is not specified. It specifies the name of a pin or port object or a collection containing a port or pin object. If you do not use a DFT signal to control the modal connections, you will be responsible to provide the appropriate test_setup proc when using the modal connections during scan or scan_retargeting DRC.

If you used a DFT signal instead, enabling the mode during the DRC will be as easy as calling the [set_static_dft_signal_values](#) command.

- **-inverse_enable**

An optional switch that is used to specify that the modal connection is to be enabled when the specified enable DFT signal or source node is low instead of high. You cannot use this option when making a connection involving an auxiliary_enable_pin because all auxiliary enable pins must be driven low during functional mode otherwise the functional mode will be broken. Also, you can only used this switch when there is only one other modal connection made to the port and this other connection is enabled by the same enable source but without the inversion.

- **-pipeline_stages *pipeline_stages***

An optional switch used to specify an integer to request between 1 to 5 levels of pipelining stages along the modal DFT connection. If you broadcast one input port to multiple destinations in a given mode, the number of pipelining stages must be equal in all broadcasted connections.

- **-nonscan**

An optional switch used to specify that the modal connection is for a non scan-based test mode. The DFT signal scan_en is not used to gate off the auxiliary enable pin when those modes are used.

Examples

Example 1

The following example has two identical instances of a module called “corea”. The gpio[3:0] were equipped with auxiliary input and output logic during Boundary Scan implementation using this syntax:

```
set spec [create_dft_spec]
read_config_data -in ${spec}/BoundaryScan -from_string {
    AuxiliaryInputOutputPorts {
        auxiliary_input_ports : gpio[1:0];
        auxiliary_output_ports : gpio[3:2];
    }
}
```

When it is time to insert the top level EDT controller, the channels are connected normally to the auxiliary input and output pins of the ports. See how the [get_auxiliary_pins](#) command is used to get the auxiliary data pins created by the Boundary Scan step. The use of the “*” symbol in the edt mode commands specifies to preserve the connections that were done when the top

level EDT controller was instantiated. The retargeting1 and retargeting2 modes each retarget the edt mode corea_i1 and corea_i2 through the gpio[2:0] ports. The retargeting3 mode broadcasts the gpio[1:0] auxiliary data inputs to both instances and uses gpio[2] for the EDT output of corea_i1 and gpio[3] for corea_i2.

```

set mode edt_mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes * \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
  -output_data_source_nodes * \
  -enable_dft_signal $mode

set mode retargeting1_mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
  -output_data_source_nodes corea_i1/edt_channels_out[0] \
  -enable_dft_signal $mode

set mode retargeting2_mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
  -output_data_source_nodes corea_i2/edt_channels_out[0] \
  -enable_dft_signal $mode

set mode retargeting3_mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
  -output_data_source_nodes corea_i1/edt_channels_out[0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[3] \
  -output_data_source_nodes corea_i2/edt_channels_out[0] \
  -enable_dft_signal $mode

```

```

check_design_rules
set spec [create_dft_spec -sri_sib_list {occ edt}]
read_config_data -in_wrapper $spec -from_string {
    Occ {
        ijtag_host_interface : Sib(occ);
        Controller(clka) {
            clock_intercept_node : clka_buf/A;
        }
    }
    Edt {
        ijtag_host_interface : Sib(edt);
        Controller(c1) {
            longest_chain_range      : 10, 50;
            scan_chain_count         : 40;
            input_channel_count      : 2;
            output_channel_count     : 1;
            connect_bscan_segments_to_lsb_chains : on;
            Connections {
                EdtChannelsIn(1) {
                }
                EdtChannelsIn(2) {
                }
                EdtChannelsOut(1) {
                    Pipelinestage {
                    }
                }
            }
        }
    }
}

set_config_value port_pin_name \
-in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(1) \
[get_single_name [get_auxiliary_pins gpio[0] -direction input]]

set_config_value port_pin_name \
-in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(2) \
[get_single_name [get_auxiliary_pins gpio[1] -direction input]]

set_config_value port_pin_name \
-in $spec/Edt/Controller(c1)/Connections/EdtChannelsOut(1) \
[get_single_name [get_auxiliary_pins gpio[2] -direction output]]

```

Following is the output created by the report_dft_modal_connections when running on the above example:

```

Dft Modal Connections
=====
Modes
-----
edt_mode           : DFT signal with usage scan_mode(unwrapped)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'

```

```

retargeting1_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
  Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
  Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
  Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'

retargeting2_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
  Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
  Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
  Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
  Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
  Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connections (2):
  Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
  Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'

Input connections
-----
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
  edt_mode      : <Existing destination>
  retargeting1_mode : corea_i1/edt_channels_in[1]
  retargeting2_mode : corea_i2/edt_channels_in[1]
  retargeting3_mode : corea_i1/edt_channels_in[1]
                : corea_i2/edt_channels_in[1]

2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
  edt_mode      : <Existing destination>
  retargeting1_mode : corea_i1/edt_channels_in[0]
  retargeting2_mode : corea_i2/edt_channels_in[0]
  retargeting3_mode : corea_i1/edt_channels_in[0]
                : corea_i2/edt_channels_in[0]

Output connections
-----
1) Port 'gpio[2]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_2_AuxOut'
  edt_mode      : <Existing source>
  retargeting1_mode : corea_i1/edt_channels_out[0]
  retargeting2_mode : corea_i2/edt_channels_out[0]
  retargeting3_mode : corea_i1/edt_channels_out[0]

2) Port 'gpio[3]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_3_AuxOut'
  retargeting3_mode : corea_i2/edt_channels_out[0]

```

add_dft_signals

Context: dft

Mode: setup

Adds static and dynamic DFT signals used to control various aspect of the DFT logic.

Usage

```
add_dft_signals {name ...}  
  [-source_nodes pin_port_spec [-make_ijtag_port]  
   | -create_from_other_signals [-parent_instance parent_instance]  
   | [-leaf_instance_name leaf_instance_name] | -create_with_tdr]
```

Description

A command used to request the addition of static and dynamic DFT signals used to control various aspect of the DFT logic. There are several pre-registered DFT signals that you can use to control the circuit during its multiple test modes.

- [Table 3-10](#) on page 190 lists the pre-registered static DFT signals.
- [Table 3-11](#) on page 209 lists the pre-registered dynamic DFT signals.

Use the [report_dft_signal_names](#) command to report the list of registered DFT signals. Use the [register_static_dft_signal_names](#) command to register additional signals.

The static DFT signals are separated into the following groups:

- **global_dft_control** — Contains signals used to control global resources such as clocking and power management circuitry.
- **logic_test_control** — Defines the signals used to configure various aspect of the circuit during logic test modes.
- **scan_mode** — Defines the signals used to configure the scan chains into the various scan configurations during logic test modes.

The logic_test_control signals include ltest_en, which is the signal used to place the circuit in logic test mode. This signal is systematically set to 1 when running any logic test mode. The int_ltest_en and the ext_ltest_en signals are used to place the circuit in internal or external mode. A scan mode that includes both the wrapper and internal scan chains of a core can be operated in internal mode by keeping int_ltest_en to 1. The same scan chain configuration can be turned into an unwrapped mode by forcing int_ltest_en to 0. See the [set_static_dft_signal_values](#) command for more information about how to set them.

The memory_bypass_en signal is another example of a logic_test_control signal. When set to 1, the tool runs ATPG on a given scan mode with the memory bypass logic enabled. This

configuration is used to cover the majority of the faults. When set to a 0, the tool keeps the memory active during scan test to cover the memory port faults using multi-load patterns.

See [Table 3-10](#) on page 190 for a detailed description of each signal and when to use them.

DFT Signals and IJTAG

By default, the static DFT signals are implemented with a special IJTAG TDR. When you specify a static DFT signal, the tool automatically creates the signal on a TDR. If the DFT signal was added in a previous insertion pass, it is automatically reused.

When the [Sib\(sti\)](#) is already present and its input that are to be connected to a DFT signal still have the `has_functional_source` attribute = 0, then the tool makes the port connections as the signals are added using this command. Refer to [Figure 10-11](#) on page 3267 and [Figure 10-12](#) on page 3268 for more information about the Sib(sti) and its interaction with logic test. See also “[How the IjtagNetwork specification is created](#)” on page 478 for further information on Sib(sti).

If you have your own IJTAG node to source these signals, you can add attributes in the `DataOutPort` wrapper of its ICL description to identify them; the signals will automatically be recognized as existing DFT signals. See the `tessent_dft_signal_*` attributes in “[icl_port](#)” on page 3126 for a description of the ICL Port Attributes that you must use to identify DFT signals on your IJTAG node. “[Example 2](#)” on page 217 shows an example of such an ICL module.

DFT Signal Creation

The DFT signals are created when running the [process_dft_specification](#) command. You use [create_dft_specification](#) to create the [DftSpecification](#).

When creating static DFT signals, the `IjtagNetwork` wrapper is present and contains a `Tdr(sri_ctrl)` wrapper with the `tessent_dft` function attribute set to “`scan_resource_instrument_dft_control`”. Do not add or delete this wrapper. The wrapper’s content is auto adjusted by the tool to match the list of static DFT controls you specified.

When creating static DFT signals with `-create_with_tdr`, the `IjtagNetwork` wrapper is present and contains a `Tdr(sri_ctrl)` wrapper with the `tessent_dft` function attribute set to “`scan_resource_instrument_dft_control`”. Do not add or delete this wrapper. The wrapper’s content is auto adjusted by the tool to match the list of static DFT controls you specified.

The `add_dft_signals` command can only be called after the `set_design_level` command was issued, unless the design level is already imported by the `read_design` command. The `design_level` affects the default behavior of the command as described in the note below:

Note

 When the `design_level` is set to `sub_block` using the `set_design_level` command, the static DFT signals default to be created as input ports by the name of the DFT signal. Sub blocks are by definition processed independently at the RTL level but synthesized as part of their parent physical block. The DFT signals created at the boundary of the sub-block will be connected to the DFT signal by the same name added in the parent physical block. You typically will not add the `int_ltest_en` and `ext_ltest_en` DFT signals inside a sub-block as only physical blocks can be wrapped cores. If for some reason you need to add the `int_ltest_en` or `ext_ltest_en` in a sub-block, by default they will be created as ports on the sub-block and connected to the respective DFT signal at the parent level. This means that you will not be able to assert `int_ltest_en` on in the parent and off inside the child `sub_block`. If you need to do that, use the `-create_with_tdr` switch to provide an independent source inside the sub-block module.

If you are only adding dynamic DFT signals, then you must run the `process_dft_specification` command even if the `DftSpecification` contains no `IjtagNetwork` wrapper. The tool creates the missing ports and exports the DFT signals into the `.tcd` file located in the `dft_inserted_designs` directory such that the signals are automatically imported in a subsequent Tesson Shell invocation for the future DFT insertion passes and for ATPG setup.

The DFT signals you add in one insertion pass are automatically imported in the subsequent DFT insertion passes. If the location of the DFT signal is no longer present, the tool issues a warning telling you that the DFT signal could not be imported. For example, if you added `edt_clock` with `-create_from_other_signals` in `pass1`, a clock gater will be added to create the DFT signal. If you synthesized the RTL and forgot to preserve the cells with the special `persistent_cell_prefix` value, the clock gater may have changed names, and it will no longer be possible to find the signal.

Table 3-10. Pre-Registered Static DFT Signal Names

Static DFT Signal Name	Usage and Default Implementation
all_test	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 1 value_in_pre_scan_drc: 1</p> <p>A global DFT control signal that is set high during all tests including all logic test and non-logic test modes such as memoryBIST.</p> <p>This signal is used by default when using the add_dft_control_points command and the -dft_signal_source_name switch is not specified. The signal is also used to control the select of the multiplexers inserted when the add_dft_control_points command refers to a pin that already has a functional source.</p> <p>See Table 3-9 on page 170 in the add_dft_control_points command description for more details.</p>
bscan_clamp_enable	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X</p> <p>A global DFT control signal that is used to force the Boundary Scan wire called select_jtag_output to one. This allows pre-loading the boundary scan with a set of values and forcing those values on the output during any test. The connection between the Boundary Scan interface module and this signal is automatic as soon as the DFT signal is added.</p>
occ_kill_clock_en	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0</p> <p>A DFT signal used to drive the kill_clock_en port on the OCC module. See the kill_clock_mode property description in the OCC wrapper.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
output_pad_disable	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X</p> <p>A global DFT control signal that is used to force the output pad buffers in high-z using the force_disable wire of the Boundary Scan. This allows forcing the output pads in high-z during any test and save on power. The connection between the Boundary Scan interface module and this signal is automatic as soon as the DFT signal is added.</p> <p> Note: If you add this signal at the top level and you have inserted BoundaryScan with the max_segment_length_for_logictest property not equal to off, the output_pad_disable DFT signal will automatically be connected to the boundary scan interface module and force to_force_disable high when asserted. You then use the set_static_dft_signal_values output_pad_disable 1 command to tri-state the output pad buffers during logictest modes.</p>
tck_occ_en	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0</p> <p>A global DFT control signal that is used to enable the mini-OCC present inside the Sib(sti) node. The signal is automatically set to 1 when you invoke the add_core_instances command on the Sib(sti) instances.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
tck_select	<p>Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0</p> <p>A global DFT control signal that is used to control the multiplexer used to inject TCK at the base of the clock trees. This signal is not needed when using the Tessent OCC as it is already equipped with a TCK multiplexer controlled by a local TDR bit.</p>
async_set_reset_static_disable	<p>Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0</p> <p>A logic test control signal that is used to completely disable all asynchronous set and reset signals during logic test modes. This signal is combined with scan_en to create the dynamic DFT signal called <code>async_set_reset_dynamic_signal</code> as described in Table 3-11 on page 209.</p> <p>Normally, the asynchronous set and reset signals are only disabled when scan_en is high. The state of the functional circuit determines if the reset and set pins are active or not during capture cycles.</p> <p>Setting the <code>async_set_reset_static_disable</code> signal to high keeps the set and reset pins gated off even during the capture cycle. It is normally set low but can be set high using the set_static_dft_signal_values command.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
control_test_point_en	<p>Usage: Logic Test Control</p> <p>reset_value : 0</p> <p>default_value_in_all_test : 0</p> <p>value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to enable the control test point.</p> <p>During the dft -scan context, use:</p> <pre>set_test_point_insertion_options \ -control_point_enable \ [get_dft_signal control_test_point_en]</pre> <p>to use the DFT signal as the control point enable.</p>
ext_ltest_en	<p>Usage: Logic Test Control</p> <p>reset_value : 0</p> <p>default_value_in_all_test : 0</p> <p>value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to isolate the output ports of a core from the internal logic. In dft -scan context, it is used with the dynamic DFT signal called scan_en to control the isolation in external modes.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
int_ltest_en	<p>Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to isolate the input ports of a core from the internal logic. In dft -scan context, it is used with the dynamic DFT signal called scan_en to control the isolation in external modes.</p> <p> Note: If you add this signal at the top level and you have inserted BoundaryScan with the max_segment_length_for_logictest property not equal to off, the int_ltest_en DFT signal will automatically be connected to the boundary scan interface module and force to_select_jtag_input high when asserted. You then need to use the set_static_dft_signal_values int_ltest_en 1 command to use the boundary scan chain and isolate the core from the primary input ports.</p>
ltest_en	<p>Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 1</p> <p>A logic test control signal that is used to enable the logic test mode. This signal is force high during all logic test modes.</p>
mcp_bounding_en	<p>Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to enable the X-bounding circuitry caused by timing exceptions.</p> <p>In dft -scan context, use:</p> <pre>set_xbounding_options -mcp_bounding_en \ [get_dft_signal mcp_bounding_en]</pre> <p>to have the added DFT signal used by X-bounding.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
memory_bypass_en	<p>Usage: Logic Test Control</p> <p>reset_value : 0</p> <p>default_value_in_all_test : 0</p> <p>value_in_pre_scan_drc: 1</p> <p>A logic test control signal that is used to enable the isolation around the memories. This signal is set to 1 by default during logic test but can be set low using the set_static_dft_signal_values command.</p>
observe_test_point_en	<p>Usage: Logic Test Control</p> <p>reset_value : 0</p> <p>default_value_in_all_test : 0</p> <p>value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to enable the observation test points.</p> <p>In dft -scan context, use:</p> <pre>set_test_point_insertion_options \ -observe_point_enable \ [get_dft_signal observe_test_point_en]</pre> <p>to have the added DFT signal used by X-bounding.</p>
se_pipeline_en	<p>Usage: Logic Test Control</p> <p>reset_value : 0</p> <p>default_value_in_all_test : 0</p> <p>value_in_pre_scan_drc: 0</p> <p>A logic test control signal that is used to enable the pipelining of the scan enable signal in order to perform launch off shift at-speed test.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
x_bounding_en	<p>Usage: Logic Test Control reset_value : 0 default_value_in_all_test :0 value_in_pre_scan_drc: X</p> <p>A logic test control signal that is used to enable the X-bounding circuitry caused by X sources.</p> <p>In dft -scan context, use:</p> <pre>set_xbounding_options -xbounding_enable\ [get_dft_signal x_bounding_en]</pre> <p>to have the added DFT signal used by X-bounding.</p>
edt_mode	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the EDT scan configuration.</p> <p>You used this scan_mode to control the edt mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped.</p> <p>When using wrapper chains, you use int_edt_mode or int_mode instead. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
multi_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: unwrapped</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the multi-chain bypass scan configuration. You use this scan_mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped.</p> <p>When using wrapper chains, you use int_multi_mode instead. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. If you are using the bypass mode built in the EDT controller, you do not need this DFT signal. You only use this signal if you are implementing the multi-chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
promoted_cells_mode	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the promoted cells scan mode. You use this mode when you have OCCs driven by embedded PLLs inside a wrapped core and this core is instantiated below another wrapped core. In that case, you simply add this DFT signal in the lower level wrapped core during OCC insertion and use “add_scan_mode promoted_cells_mode” during scan insertion.</p> <p>Chains belonging to OCCs driven by embedded PLLs will automatically be included in this mode. They are also automatically included in all scan modes of type external. At the next level up, when doing scan insertion in the parent core, the chain associated with the promoted_cells_mode in the child cores will automatically be promoted to belong to the wrapper chains of the parent core so that the OCC is accessible from the top level using the wrapper chains of its child wrapped cores.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
single_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: unwrapped</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the single chain bypass scan mode. You used this scan_mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped.</p> <p>See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. If you are using the single chain bypass mode built in the EDT controller, you do not need this DFT signal. You only use this signal if you are implementing the single chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
int_edt_mode	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the EDT scan mode when using wrapper chains because the scan_mode_type of this signal is internal. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. If you are using the multi and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal. You only use this signal if you are implementing the single and multi chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
int_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: internal</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the internal mode when using wrapper chains and you are using the multi and single chain bypass mode built in the EDT controller. This DFT signal has scan_mode_type equal to internal and is used to configure the internal scan configuration of a wrapped core.</p> <p>See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. If you are implementing the single and multi chain bypass mode at the scan insertion stage, you will use the int_edt_mode, int_multi_mode, and int_single_mode signals.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
int_multi_mode	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the internal multi-chain scan mode when using wrapper chains. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal.</p> <p>This DFT signal has scan_mode_type equal to internal and is thus to be used to configure the internal multi-chain scan configuration of a wrapped core. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. You only use this signal if you are implementing the single and multi-chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
int_single_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: internal</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the internal single chain scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to internal and is used to configure the internal single chain scan configuration of a wrapped core. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values. If you are using the multi and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal. You only use this signal if you are implementing the single and multi chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
ext_edt_mode	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the external EDT scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to external and is used to configure the external EDT scan configuration of a wrapped core. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values.</p> <p>Typically, the wrapper chains are not driven by an EDT controller but it is an option that can be used. If you are using the multi and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT to drive your wrapper chain and you are implementing the external single and multi chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
ext_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: external</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the external scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to external and is used to configure the external scan configuration of a wrapped core. See the description of the set_static_dft_signal_values command to understand the importance of using scan_mode DFT signals with the accurate scan_mode_type values.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
ext_multi_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: external</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the external multi chain scan mode when using wrapper chains. Typically, the wrapper chain are not driven by an EDT controller, but it is an option that can be used. If you are using the multi and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT controller to drive your wrapper chain and you are implementing the external single and multi chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
ext_single_mode	<p>Usage: Scan Mode</p> <p>reset_value : 0</p> <p>scan_mode_type: external</p> <p>value_in_pre_scan_drc: X</p> <p>A scan mode signal that is used to enable the external single chain scan mode when using wrapper chains. Typically, the wrapper chain are not driven by an EDT controller, but it is an option you can use. If you are using the multi and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT to drive your wrapper chains, and you are implementing the single and multi chain bypass mode at the scan insertion stage.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-10. Pre-Registered Static DFT Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation
retargeting#_mode de = 1, 2, ..., 8	<p>Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X</p> <p>A set of scan mode signals used to configure the retargeting modes that make use of the internal scan modes of child blocks. For example: retargeting1_mode can be used to provide scan channel data to the internal scan mode of core_a, core_b, and core_c, and retargeting2_mode can be used to provide scan channel data to the internal scan mode of core_d, and core_e.</p> <p>If you have more than 8 retargeting combinations at a given level, you simply need to register more scan_mode signals of type retargeting using the register_static_dft_signal_names command.</p> <p>In dft -scan context, this signal is automatically associated to the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.</p>

Table 3-11. Pre-Registered Dynamic DFT Signal Names

Dynamic DFT Signal Name	Usage and Default Implementation
async_set_reset_dynamic_disable	<p>A DFT signal that is used to disable the asynchronous set and reset pins of scannable flip-flops during shift mode. This signal is high during the load_unload cycles and is allowed to return low in the capture cycles. When using the -create_from_other_signals switch, it is created by ORing the scan_en and the async_set_reset_static_disable DFT signals. The DFT signals scan_en and async_set_reset_static_disable must be added before the async_set_reset_dynamic_disable DFT signal can be added using the -create_from_other_signals switch.</p> <p>If you want to create this DFT signal without creating the async_set_reset_static_disable DFT signal, use the -source_node switch and map the async_set_reset_dynamic_disable DFT signal on the same node you have defined the scan_en DFT signal.</p> <p>By default, the async_set_reset_static_disable signal is set to 0 such that the async_set_reset_dynamic_disable signal follows scan_en. However, the set and reset pins can be disabled systematically by setting the async_set_reset_static_disable signal to 1 using the set_static_dft_signal_values command. One can choose to set the async_set_reset_static_disable signal to avoid D1 violations or to reduce the amount of care bits needed by ATPG to disable the set and reset signals in order to observe the fanin of the scannable flip-flops. You must, however, generate some patterns with the async_set_reset_static_disable signal set to 0 in order to gain coverage of the reset circuitry.</p>  <pre> graph LR scan_en[scan_en] --> OR(()) static_disable(async_set_reset_static_disable) --> OR OR --> dynamic_disable(async_set_reset_dynamic_disable) </pre>

Table 3-11. Pre-Registered Dynamic DFT Signal Names (cont.)

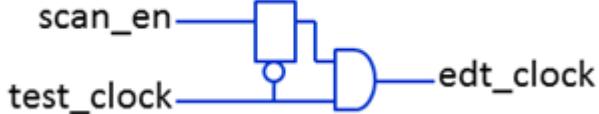
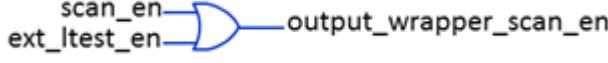
Dynamic DFT Signal Name	Usage and Default Implementation
edt_clock	<p>A DFT signal that is used to supply a clock to the EDT controller. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enable sourced by scan_en. This is the recommended method as it allows to share a common clock source for both the edt_clock and the shift_capture_clock. The DFT signal scan_en and test_clock must be added before the edt_clock can be added with the -create_from_other_signals switch. See Figure 3-10 on page 212 for an illustration of its waveforms.</p>  <p>When hybrid LogicBIST is present, this gater will be created within the LogicBIST controller. If this gater was inserted in a previous insertion pass, then it will be removed when inserting the hybrid controller and the existing connections will be moved to the LogicBIST output port.</p>
edt_update	<p>A DFT signal that is used to supply the update enable port of the EDT controller. This signal is also used to create the shift_capture_clock. See Figure 3-10 on page 212 for an illustration of its waveforms.</p>
input_wrapper_scan_en	<p>A DFT signal that is used (only for third-party scan insertion) to supply the scan_en signal to the input wrapper chains. When using the -create_from_other_signals switch, it is created by ORing scan_en with int_ltest_en.</p> 
output_wrapper_scan_en	<p>A DFT signal that is used (only for third-party scan insertion) to supply the scan_en signal to the output wrapper chains. When using the -create_from_other_signals switch, it is created by ORing scan_en with ext_ltest_en.</p> 
scan_en	<p>A DFT signal that is used to define the source of scan_en. When using an OCC, this signal is constrained off during the capture cycle. When the async_set_reset_static_disable signal is not set to 1, you must make sure the scan_en source is high during all cycles of the load_unload procedure to avoid D1 violations. See Figure 3-10 on page 212 for an illustration of its waveforms.</p>

Table 3-11. Pre-Registered Dynamic DFT Signal Names (cont.)

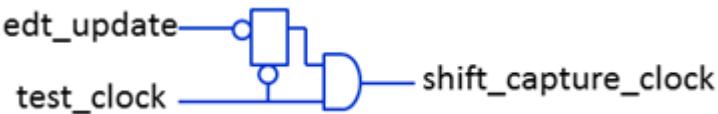
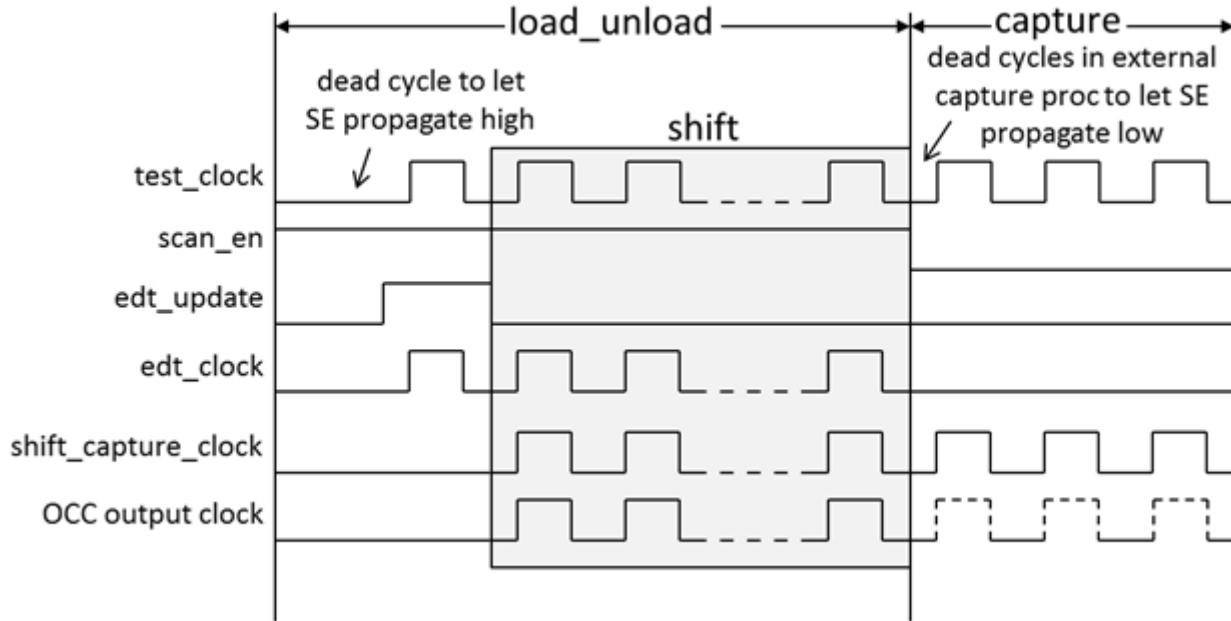
Dynamic DFT Signal Name	Usage and Default Implementation
shift_capture_clock	<p>A DFT signal that is used to supply the slow clock to the OCC controller. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enables sourced by edt_update. This is the recommended method as it allows to share a common clock source for both the edt_clock and the shift_capture_clock. The DFT signal edt_update and test_clock must be added before the shift_capture_clock can be added with the -create_from_other_signal switch. See Figure 3-10 on page 212 for an illustration of its waveforms.</p>  <p>When hybrid LogicBIST is present, this gater will be created within the LogicBIST controller. If this gater was inserted in a previous insertion pass, then it will be removed when inserting the hybrid controller and the existing connections will be moved to the LogicBIST output port.</p>
test_clock	<p>A DFT signal that defines a clock input as the test_clock, which is to be used as the common source to synthesize the edt_clock and the shift_capture_clock. You do not need this signal if the edt_clock and the shift_capture_clock are sourced by independent sources. See Figure 3-10 on page 212 for an illustration of its waveforms.</p> <p> Note: You cannot share or reuse the test_clock with an already-existing functional clock that feeds sequential elements intended to be tested with ATPG. The test_clock needs to be a separate independent clock because the test_clock must pulse an extra amount of time to initialize the EDT without disturbing the functional clock domains.</p>

Figure 3-10. Waveforms of Selected Dynamic DFT Signals



Knowing which Signals to Add for Logic Test

Follow these instructions to know which signals to add for logic test:

- When you have memory BIST:

```
add_dft_signals memory_bypass_en
```

- When you have at least one Scan Tested Instrument like memory BIST or boundary scan (when boundary scan segments will be used for logic test):

```
add_dft_signals tck_occ_en
```

- When you are using logic test:

```
add_dft_signals ltest_en async_set_reset_static_disable  
add_dft_signals scan_en -source_nodes my_scan_en
```

- When you are creating a wrapped core:

```
add_dft_signals int_ltest_en ext_ltest_en  
add_dft_signals int_mode ext_mode
```

- When using EDT, create the shift_capture_clock and edt_clock from a common test_clock:

```
add_dft_signals test_clock edt_update -source_nodes \  
{my_test_clock my_edt_update}  
add_dft_signals shift_capture_clock edt_clock \  
-create_from_other_signals
```

- When not using EDT:

```
add_dft_signals shift_capture_clock -source_nodes my_test_clock
```

- When reusing boundary scan for logic test at the top level, and you want to be able to perform logic test without contacting inputs that have boundary scan cells:

```
add_dft_signals int_ltest_en output_pad_disable
```

Arguments

- *name* ...

A required repeatable string value used to specify one or more names of DFT signals to add. You can specify several names in one command line invocation. You can also supply several names in a single well formatted Tcl list. The name values must correspond to built in DFT signal names or user registered ones. See the [register_static_dft_signal_names](#) command description for instructions on how to register your own.

- *-source_nodes pin_port_net_spec*

An optional switch that is used to associate ports, pins, or nets to the DFT signal. This option is mandatory when adding the following dynamic DFT signals: scan_en, test_clock, and edt_update. Specified pin or net names must refer to existing objects. Port names are allowed to refer to ports that are to be created as long as they are scalar names and the design level set using the [set_design_level](#) command is not “chip”. When the design level is set to “chip”, specified port names must refer to existing ports and the ports must be connected to a properly described pad buffer cell. The connection to the DFT signal will automatically map the specified top-level port to the output pin of its associated input pad buffer.

If you have added auxiliary input logic on the input pad used to source the dynamic DFT signals such as scan_en, test_clock and edt_clock, the DFT signal will automatically be mapped on the auxiliary input data pin and an

“add_dft_control_point -dft_signal_source_name ltest_en” command will be inferred to control the auxiliary input enable pin as shown in Example 5. See the documentation of the [get_auxiliary_pins](#) and the [add_dft_modal_connections](#) commands for a detailed description of the auxiliary logic and how to describe third party implementation using a Tcl dictionary called “::auxiliary_data_dict”.

If you do not specify any of the following switches:

- *-create_from_other_signals*
- *-create_with_tdr*
- *-source_node*

the *-create_with_tdr* switch is assumed when the *design_level* is not a *sub_block*; otherwise the *-source_node* switch with a port name equal to the DFT signal name is inferred by the tool. This is the recommended implementation for all static DFT signals. See the [set_static_dft_signal_values](#) command to see how the setting of the static DFT signals is automated when sourced by IJTAG.

- **-make_ijtag_port**

An optional switch that can be used when adding a static DFT signal with the `-source_nodes` switch referring to ports. The ports will be identified as [DataInPorts](#) when running [extract_icl](#). When running ICL extract at the next level, those ports will be verified to be properly sourced by IJTAG data ports. The [set_static_dft_signal_values](#) command remains usable for setting those static DFT signals specified on a port when you also specify the `-make_ijtag_port` switch. The switch is inferred automatically when the static DFT signal is defined on a port and the `design_level` was specified as “chip” using the [set_design_level](#) command. It is also inferred when the `-source_node` switch is inferred to be equal to the DFT signal name when the `design_level` was specified as “sub_block” using the [set_design_level](#) command. See [Example 4](#).

- **-create_from_other_signals**

An optional switch that specifies to create the DFT signal from other signals. The list of the DFT signals that can be created from other signals are `edt_clock`, `shift_capture_clock`, `async_set_reset_dynamic_disable`, `input_wrapper_scan_en`, and `output_wrapper_scan_enable`. It is actually recommended that you use the `-create_from_other_signals` switch when adding them as it make the timing closure easier during synthesis and layout.

See [Table 3-11](#) on page 209 for a description of those signals and how they are synthesized from other signals. [Figure 3-10](#) on page 212 shows their respective waveforms.

- **-parent_instance *parent_instance***

An optional switch-value pair that specifies an instance in which to instantiate the clock gating cell used to synthesize the `edt_clock` and the `shift_capture_clock` signals. When left unspecified, the parent instance defaults to the current design. You can create several copies of the clock gating cells as long as you specify a different parent instance for each one of them. The other three dynamic signals that can be created from other signals only have one copy. The OR gate is instantiated in the parent instance of the static DFT signal that feeds one input of the OR gate.

- **-leaf_instance_name *leaf_instance_name***

An optional switch-value pair that specifies the leaf name of the logic gate used to synthesize the logic signal. By default, the leaf instance name is the name of the DFT signal prefixed by the value specified with the `persistent_cell_prefix` property inside the [DftSpecification](#) wrapper. The default is “`tessent_persistent_cell_`”.

- **-create_with_tdr**

An optional switch that specifies that the DFT signal is to be created from a TDR. This switch is assumed when the `-source_node` and the `-create_from_other_signals` switches are not specified. This switch is only allowed when adding static DFT signals.

This switch is assumed when the `-source_node` and the `-create_from_other_signals` switches are not specified or inferred. See the `-source_node` switch description to know when it is inferred automatically.

Examples

Example 1

The following example defines the DFT signals needed in the given block. The block is implementing MemoryBIST and Logic Test with EDT and OCC. The signals are added in the first insertion pass where memoryBIST and IJTAG are inserted in the RTL. The logic test connection between the Sib(sti) and DFT signal is automated and happens in Pass2 when they are added. The second dofile inserts the DFT signal needed by the logic test flow and inserts the EDT and OCC controllers. Finally, the third dofile shows how the scan_mode signals are specified as the enable signals of the scan_modes.

RTL Pass1

```
set_context dft -rtl
read_cell_library ...
read_verilog ...
set_current_design my_core1
set_design_level physical_block
set_dft_specification_requirements -memory_test on

add_dft_signals ltest_en int_ltest_en ext_ltest_en \
    memory_bypass_en async_set_reset_static_disable \
    int_mode ext_mode
add_dft_signals scan_en test_clock edt_update \
    -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
    input_wrapper_scan_en output_wrapper_scan_en \
    async_set_reset_dynamic_disable \
    -create_from_other_signals

...
```

RTL Pass2

```
set_context dft -rtl2
read_cell_library ...
read_design my_core1 -design_id rtl
set_current_design my_core1
check_design_rules
set spec [create_dft_specification -sib_sri_list {occ edt}]
read_config_data -in $spec -from_string {
    OCC {
        ijttag_host_interface : Sib(occ);
    }
    EDT {
        ijttag_host_interface : sib(edt) ;
    }
}
set edt1 [add_config_element EDT/Controller(1)      -in $spec]
set_config_value longest_chain_range {100 200}      -in $edt1
set_config_value scan_chain_count 500              -in $edt1
set_config_value input_channel_count 8             -in $edt1
set_config_value input_channel_count 3             -in $edt1

set_config_value EDT/Connections/edt_clock -in $spec \
    DftSignal(edt_clock)
set_config_value EDT/Connections/edt_update -in $spec \
    DftSignal(edt_update)

add_dft_signals ltest_en int_ltest_en ext_ltest_en \
    memory_bypass_en async_set_reset_static_disable \
    int_mode ext_mode
add_dft_signals scan_en test_clock edt_update \
    -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
    input_wrapper_scan_en output_wrapper_scan_en \
    async_set_reset_dynamic_disable \
    -create_from_other_signals

...
foreach clock [get_clocks] {
    set id [get_clock_option $clock -label]
    set occ [add_config_element OCC/Controller($id) -in $spec]
    set_config_value clock_intercept_node -in $occ \
        [get_single_name $clock]
    set_config_value Connections/slow_clock -in $occ \
        DftSignal(shift_capture_clock)
    set_config_value Connections/scan_en -in $occ \
        DftSignal(scan_en)
}
...
```

Scan Insertion Pass

```

set_context dft -scan
read_cell_library ...
read_verilog ...
read_design my_core1 -design_id rtl2 -no_hdl
set_current_design my_core1

set edt_icl_module [get_icl_module \
    -filter tessent_instrument_type==mentor::edt]
set edt_instance [lindex \
    [get_attribute_value_list \
        [get_icl_instance -of_module $edt_icl_module] \
        -name hierarchical_design_instance] 0]

analyze_wrapper_cells
add_scan_mode int_mode \
    -type internal \
    -chain_length 200 \
    -edt_instance $edt_instance \
    -enable_dft_signal int_mode
add_scan_mode ext_mode \
    -type external \
    -chain_length 200

```

Example 2

This example shows the DFT signals pre-existing on a third-party TDR. The first part shows the DataOutPort attributed in the ICL module. See the `tessent_dft_signal_*` attributes description in the [icl_port](#) data model section. The second part calls the [report_dft_signals](#) command once the design has been elaborated and the ICL module matched to its Verilog counterpart to show how the DFT signals were auto detected and added as existing signals.

Section of MyIJTAGNode.icl

```

Module MyIJTAGNode {
    ...
    DataOutPort XXX_all_test {
        Source TDR[0];
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tessent_dft_signal_name = "all_test";
        Attribute tessent_dft_signal_usage = "global_dft_control";
        Attribute tessent_dft_signal_default_value_in_all_test = 1;
        Attribute tessent_dft_signal_reset_value = 0;
    }
    DataOutPort XXX_ltest_en {
        Source TDR[3];
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tessent_dft_signal_name = "ltest_en";
        Attribute tessent_dft_signal_usage = "logic_test_control";
        Attribute tessent_dft_signal_value_in_pre_scan_drc = "1";
        Attribute tessent_dft_signal_reset_value = 0;
    }
    DataOutPort XXX_int_mode {
        Source TDR[6];
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tessent_dft_signal_name = "int_mode";
        Attribute tessent_dft_signal_usage = "scan_mode";
        Attribute tessent_dft_signal_scan_mode_type = "internal";
        Attribute tessent_dft_signal_reset_value = 0;
    }
    DataOutPort XXX_ext_mode {
        Source TDR[7];
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tessent_dft_signal_name = "ext_mode";
        Attribute tessent_dft_signal_usage = "scan_mode";
        Attribute tessent_dft_signal_scan_mode_type = "retargeting";
        Attribute tessent_dft_signal_reset_value = 0;
    }
    ScanRegister TDR[7:0] {
        ScanInSource si;
        ResetValue 8'b0;
    }
    ...
}

```

Dofile Section

```

set_context dft -rtl
read_cell_library ...
read_verilog ...
set_current_design my_core1
report_dft_signals

// -----
// Name      Status   Origin          Type           Location
// -----
// all_test  Existing  ICL attributes IJTAG register tdr/XXX_all_test
// ltest_en  Existing  ICL attributes IJTAG register tdr/XXX_ltest_en
// int_mode  Existing  ICL attributes IJTAG register tdr/XXX_int_mode
// ext_mode  Existing  ICL attributes IJTAG register tdr/XXX_ext_mode

```

Example 3

This example shows you which DFT signal you need to add and when:

```
# When you have memory BIST
add_dft_signals memory_bypass_en

# When you have at least one Scan Tested Instrument like Memory BIST
add_dft_signals tck_occ_en

# When you are using logictest
add_dft_signals ltest_en
add_dft_signals scan_en -source_node my_scan_en

# When you are building a wrapped core
add_dft_signals int_ltest_en ext_ltest_en int_mode ext_mode

# When using EDT

add_dft_signals test_clock edt_update \
    -source_node {my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
    -create_from_other_signals

# When not using EDT
add_dft_signals shift_capture_clock -source_node {my_test_clock}
report_dft_signals
```

```
// Existing and to be created DFT signals
// =====
// -----      -----
// Name        Status   Origin    Type       Location
// -----      -----
// edt_clock   To be   User      ClockGateAnd
//             created  added
//                               <persistent_cell_prefix>
//                               edt_clock
// edt_update  To be   User      Port       my_edt_update
//             created  added
// ext_ltest_en To be   User      IJTAG      register
//                 created  added
// ext_mode    To be   User      IJTAG      register
//                 created  added
// int_ltest_en To be   User      IJTAG      register
//                 created  added
// int_mode   To be   User      IJTAG      register
//                 created  added
// ltest_en   To be   User      IJTAG      register
//                 created  added
// memory_bypass_en To be   User      IJTAG      register
//                 created  added
// scan_en    To be   User      Port       my_scan_en
//                 created  added
// shift_capture_clock To be   User      ClockGateAnd
//                 created  added
//                               <persistent_cell_prefix>
//                               shift_capture_clock
// tck_occ_en  To be   User      IJTAG      register
//                 created  added
// test_clock  To be   User      Port       my_test_clock
//                 created  added
//
```

Example 4

The following example illustrates how the `-make_ijtag_port` switch is automatically inferred for the static DFT signals defined on ports when the *design_level* is chip. In this example, the node `gpio[136]` (in red) is the inferred `ijtag_port`.

```
add_dft_signals test_clock edt_update ltest_en \
  -source_node {gpio[135] gpio[134] gpio[136]}
report_dft_signals

// Existing and to be created DFT signals
// =====
// -----      -----
// Name        Status   Origin    Type       Location
// -----      -----
// edt_update  Existing  User added  Port       gpio[134]  -> gpio134/C
// ltest_en   Existing  User added  IJTAG port  gpio[136]  -> gpio136/C
// test_clock Existing  User added  Port       gpio[135]  -> gpio135/C
```

Example 5

The following example uses `add_dft_signal` to declare the source of `scan_en`, `test_clock` and `edt_update` DFT signals. The pads are equipped with auxiliary data logic using the [AuxiliaryInputOutputPorts](#) wrapper during boundary scan insertion, so they are automatically mapped to the auxiliary input pin and the auxiliary input enable pins are inferred to be driven by the “`ltest_en`” DFT signal.

```
add_dft_signals scan_en test_clock edt_update -source_node {gpio[32:30]}
// Note: Adding DFT signal 'ltest_en' with the -create_with_tdr option.

// Note: Inferred 'add_dft_control_points
// top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_32_AuxInEn
// -dft_signal_source_name ltest_en' because the DFT signal 'scan_en' is
// sourced by an auxiliary data input pin.

// Note: Inferred 'add_dft_control_points
top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_31_AuxInEn
-dft_signal_source_name ltest_en'
// because the DFT signal 'test_clock' is sourced by an auxiliary data
// input pin.

// Note: Inferred 'add_dft_control_points
top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_30_AuxInEn
-dft_signal_source_name ltest_en'
// because the DFT signal 'edt_update' is sourced by an auxiliary data
// input pin.
```

report_dft_signals// Existing and to be created DFT signals

```
// =====
// -----  -----
// Name      Status   Origin    Type       Location
// -----  -----
// edt_update Existing  User added Port      gpio[30] -> DEF_inst/gpio_30_AuxIn
// ext_ltest_en Existing User added IJTAG port gpio[33] -> gpio33/C
// ltest_en   To be created User added IJTAG register
// scan_en    Existing  User added Port      gpio[32] -> DEF_inst/gpio_32_AuxIn
// test_clock Existing  User added Port      gpio[31] -> DEF_inst/gpio_31_AuxIn
```

Related Topics

[add_dft_control_points](#)
[add_dft_clock_mux](#)
[delete_dft_signals](#)
[get_dft_signal](#)
[process_dft_specification](#)
[register_static_dft_signal_names](#)
[report_dft_signals](#)
[report_dft_signal_names](#)

[set_static_dft_signal_values](#)

add_display_callout

Context: all contexts

Mode: setup, analysis

Displays a text string in a callout box on a specified object in the schematic window.

Usage

```
add_display_callout object_spec -string string  
[-DIsplay {FLat_schematic | HIErarchical_schematic}]
```

Description

Displays a user defined text string in a callout box associated with a specified object. If the object is not currently displayed, it will be displayed along with the callout box in the desired schematic window. The object and callout box will be displayed in the Flat Schematic window by default.

Note

 The callout box is not persistent. Once the object it refers to is deleted from the schematic, the callout box is removed as well. If the object is added again to the schematic, the callout box will not reappear until it is explicitly restored with `add_display_callout`.

Refer to “[Working with Attributes](#)” in the *Tessent Shell User’s Manual* for information on displaying object attributes in callout boxes. Attributes can be created, modified and have visibility in callout boxes controlled by the `register_attribute`, `set_attribute_value`, and the `set_attribute_options -display_in_gui` commands respectively.

Note

 Refer to “[Controlling the Display of Callouts in the Flat and Hierarchical Schematic Windows](#)” in the *Tessent Shell User’s Manual* for further information on display controls.

Arguments

- ***object_spec***

The value of *object_spec* can be the name of a port, pin, net, pseudo_port or instance object, a gate_id, or the name or id of a gate_pin. The value of *object_spec* can also be a Tcl list or Tcl collection of one or more names of port, pin, net, pseudo_port, instance or gate_pin objects.

If the object is a pseudo_port and -display is set to “flat_schematic”, the pseudo_port is drawn as a port. If -display is set to “hierarchical_schematic”, the mapping pins of the pseudo_ports are displayed in the schematic.

- **-string *string***

The value of *string* needs to be enclosed within double quotes, and specifies the text that will be displayed in the callout box. If there is an existing callout box on the object, then the

new string will be appended into the existing callout box. A golden text background color is implemented by add_display_callout to distinguish it from other callout box text that may be displayed.

- -DIsplay {FLat_schematic | HIEarchical_schematic}

An optional switch and literal pair that specifies the schematic window to which the tool should add the specified object and callout box. By default, they will be displayed in the Flat Schematic window if the -display argument is not provided.

Examples

Example 1

The following example adds the text string “Object text” to all the instance names with a prefix of reg_d_1 or reg_d_2 and displays them in the Flat Schematic window:

```
add_display_callout {reg_d_1* reg_d_2*} -string "Object text"
```

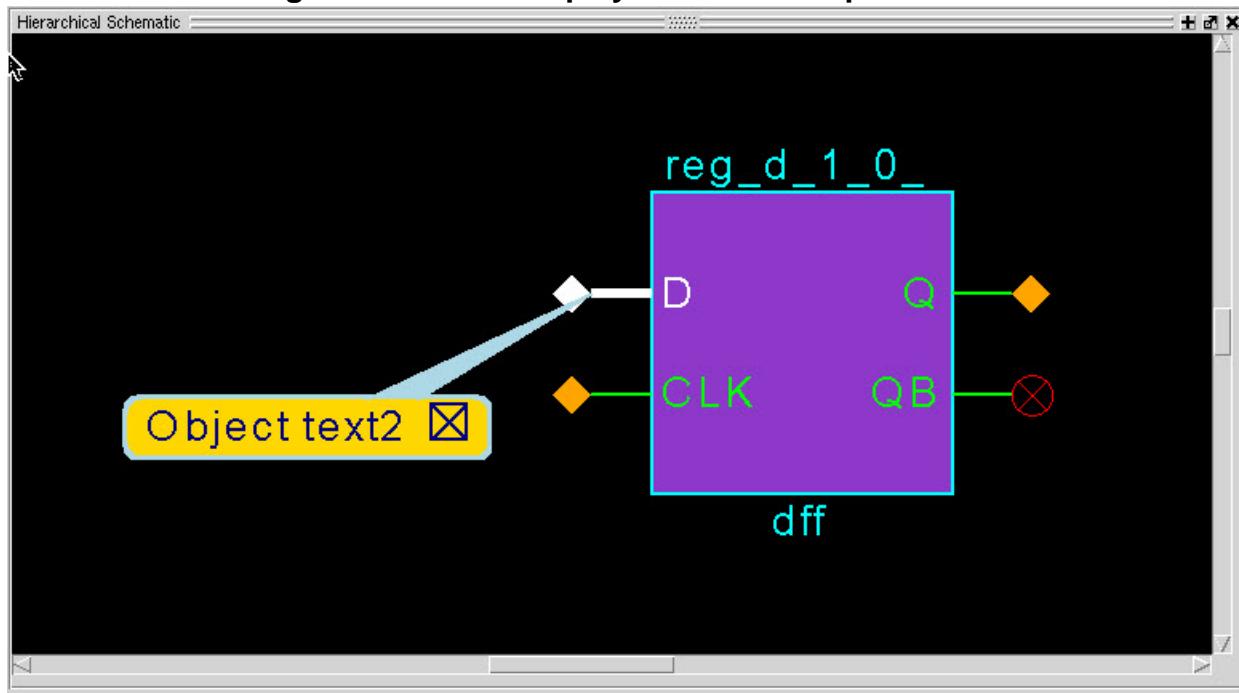
Example 2

The following example adds the text string “Object text2” to the callout box for the D input of instance reg_d_1_0_ and displays the instance and callout box in the Hierarchical Schematic window:

```
add_display_callout reg_d_1_0_/D -string "Object text2" -display hierarchical_schematic
```

The resulting schematic is shown in [Figure 3-11](#) below:

Figure 3-11. add_display_callout Example Result



Related Topics

[delete_display_callout](#)

[add_display_data](#)

Context: all contexts

Mode: setup, analysis

Adds specified data columns to the Data window of DFTVisualizer, opening the Data window if it is not already open. If data corresponding to an added column is available, it is displayed.

Usage

```
add_display_data {  
    Error_pattern | PATtern_index pattern_index [-Internal | -External] |  
    Parallel_pattern pattern_number (0..63) | Fault_status | TIE_value |  
    COnstrain_value | Seq_depth_data | Clock_cone pin_name |  
    CAPTURE_PROCedure procedure_name |  
    Drc_pattern {{Test_setup [{-{Cycle | -Time} n1} [n2 | End]]}} | Load_unload | SHift |  
        SKew_load | SHADOW_Control | Master_observe |  
        SHADOW_Observable | STate_stability | TEST_End [time | -All]} }
```

Note

 The following DFTVisualizer menu paths are available when the associated data or procedure is available; so you may not see all these selections all of the time.

Possible DFTVisualizer Menu Paths:

Data > Capture Procedure | Test-Setup | Load-Unload | Shift | State-Stability | Error Pattern | Pattern Index | Parallel Pattern | Fault | Tied | Constrain | Sequential Depth | Clock Cone | Test-End

The following arguments are not supported in dft -scan: Pattern_index, Parallel_pattern, Fault_status, and Seq_depth_data.

Description

Adds specified data columns to the Data window of DFTVisualizer, opening the Data window if it is not already open. If data corresponding to an added column is available, it is displayed.

The command's arguments and usage are based on the set_gate_report command and control the types of data displayed similar to how the set_gate_report command controls the data output of the report_gates command.

If a column has been added and then removed during a tool session, you can re-add it by moving the cursor to the column title bar, then using the right mouse button popup menu to reselect the column for display.

Use the [delete_display_data](#) command (or the right mouse button popup menu) to remove data columns you no longer need.

Arguments

For complete detailed descriptions of the type of data that the arguments control, refer to the similar arguments listed in the [set_gate_report](#) command description.

Examples

The following example adds a column displaying the simulated values for cycles 5 through 10 of the test_setup procedure:

```
add_display_data drc_pattern test_setup -cycle 5 10
```

Related Topics

[delete_display_data](#)

[report_display_instances](#)

[report_gates](#)

[set_gate_report](#)

add_display_instances

Context: all contexts

Mode: setup, analysis

Displays instances in DFTVisualizer and enables you to trace visually through the design using the Flat Schematic or Hierarchical Schematic window.

Usage

```
add_display_instances { object_spec
    [ -Backward [-One | -End_point] |
     -Forward [-One | -All | -End_point] |
     -Up | -DOwn]
    | -PAth { start end } }
    [-DIsplay {FLAt_schematic | HIEarchical_schematic | DAta | WAve }]
```

Description

Displays instances in DFTVisualizer and enables you to trace visually through the design using the Flat Schematic or Hierarchical Schematic window.

The add_display_instances command has two purposes:

- Adds display instances to DFTVisualizer windows

When you add an instance to the Flat Schematic window, DFTVisualizer updates the display differently depending on the gate level that is in effect, as summarized in the following table:

Table 3-12. How the DFTVisualizer Flat Schematic Window Displays Instances

Gate Level of Instance You Specify	“set_gate_level Design” in effect	“set_gate_level Primitive” in effect
Primitive	Displays the design level instance of which the primitive is a part	Displays the primitive
Design	Displays the design level instance	Displays all the primitives in the design level instance

When you add a root instance to the Hierarchical Schematic window from the Browser using the popup **View In > <window_name>** menu item, the tool issues the “add_display_instances -display hierarchical_schematic -down” command which displays the top level of the design with all of the major design blocks one hierarchical level below the top level and their interconnecting nets; the top-level ports are represented by port symbols. If you want to only see the top-level port symbols instead

of the expanded root design, you can do this by using the “add_display_instances - display_hierarchical_schematic” command.

- Traces visually through the design

Tracing is the process of displaying additional gates that are connected to an already displayed gate. You might want to trace in order to expand the schematic view, gain insight into circuit behavior, or get oriented within a large design. In the Flat Schematic window, you can trace from a displayed instance forward towards the primary output pins or backward towards the primary input pins at the gate level currently specified by the set_gate_level command. In the Hierarchical Schematic window, you can additionally trace from a displayed instance up or down the design hierarchy.

Arguments

- *object_spec*

The value of object_spec can be the name of a port, pin, net, pseudo_port or instance object, a gate_id, or the name or id of a gate_pin. The value of object_spec can also be a Tcl list or Tcl collection of one or more names of port, pin, net, pseudo_port, instance or gate_pin objects.

If the object is a pseudo_port and -display is set to “flat_schematic”, the pseudo_port is drawn as a port. If -display is set to “hierarchical_schematic”, the mapping pins of the pseudo_ports are displayed in the schematic.

- -Backward

An optional switch that specifies to trace from the given instances backward, towards the primary input pins.

If you do not explicitly specify a stopping point switch (-One, -End_point) in combination with this switch, the command default is for the backward trace to include only *one* level of gates (Flat Schematic window) or one hierarchical instance (Hierarchical Schematic window).

- -Forward

An optional switch that specifies to trace from the given instances forward, towards the primary output pins.

If you do not explicitly specify a stopping point switch (-One, -End_point, or -All) in combination with this switch, the command default is for the forward trace to include only *one* level of gates (Flat Schematic window) or one hierarchical instance (Hierarchical Schematic window).

- -Up

An optional switch that specifies for DFTVisualizer to trace up one level of hierarchy from the given instance(s). This switch is not valid for the Flat Schematic window.

- **-DOwn**

An optional switch that specifies for DFTVisualizer to trace down one level of hierarchy from the given instance(s). This switch is not valid for the Flat Schematic window.

Each of the following optional switches determine the stopping point for a trace by specifying the last gate you want DFTVisualizer to include in the display. If used, select just one to include in the command string.

- **-One**

An optional switch whose behavior depends on whether you are doing a backward trace or a forward trace, as follows:

Backward trace — Display only the first gate backward from a specified input pin (or from each input pin of a specified instance) from which you are tracing. This is the default for a backward trace.

Forward trace — Display only one fanout gate connected to a specified output pin (or connected to each output pin of a specified instance) from which you are tracing, even if the pin fans out to multiple gates.

This switch must be accompanied by the **-Forward** or **-Backward** switch.

- **-End_point**

An optional switch that specifies to continue the trace until reaching either a primary input, primary output, tie gate, memory, or sequential element (latch or flip-flop, scan or nonscan). This switch must be accompanied by the **-Forward** or **-Backward** switch.

This switch is not supported in the Hierarchical Schematic window.

- **-All**

An optional switch that, together with the **-Forward** switch, specifies to trace to all the fanout gates connected to an output pin from which you are tracing. This switch must be accompanied by the **-Forward** switch.

- **-PAth *start end***

A repeatable switch and two-string triplet that specifies to display all paths in the circuit beginning at the *start* pin and ending at the *end* pin. For a Flat Schematic, a *gate_pin* would be specified, such as 53.0, and for a Hierarchical Schematic a *pin_pathname* would be specified.

Note

 When this switch is specified in the Hierarchical Schematic window, the tool attempts to trace the path in one direction beginning at *start* pin; the tool returns an error if *end* pin is not found. In the Flat Schematic window, the tool first attempts to trace the path in the direction specified by the user. If that path is not found, the tool then attempts to trace the path in the opposite direction. If that path is not found, the tool returns an error message.

- -Display *window_name*

An optional switch and repeatable literal that specify the DFTVisualizer windows to which the tool should add instances. The valid options for the *window_name* argument, from which you can select as many as you want, are as follows:

FLAt_schematic

A literal that specifies the Flat Schematic window.

HIEarchical_schematic

A literal that specifies the Hierarchical Schematic window.

DAta

A literal that specifies the Data window.

WAve

A literal that specifies the Wave window.

The default is to add instances to the Flat Schematic window only.

Examples

The following example opens DFTVisualizer if not already open and displays a single gate in the Flat Schematic window by specifying the gate identification number (51):

add_display_instances 51

The next example specifies for the tool to additionally display all the gates in the first level of fanout from the Q output of gate 51. Gates connected directly to the Q output comprise the first level of fanout gates. (Gates fed by the first level of fanout gates comprise the second level of fanout gates and are not displayed.)

add_display_instances 51.0 -forward -all

The next example clears the schematic display of all gates, then creates a new display that shows the specified gate, along with a backtracking of all the gates until the trace reaches either a primary input or tie gate.

**delete_display_instances -all
add_display_instances 51 -backward -end_point**

Related Topics

[delete_display_instances](#)

[read_modelfile](#)

[report_display_instances](#)

[set_gate_level](#)

add_display_path

Context: all contexts

Mode: setup, analysis

Displays specified objects or paths in the schematic window.

Usage

```
add_display_path object_spec
  {-direction {forward | backward}
   [-Highlight {0|1|2|3|4|5|none | green | blue | orange | yellow | red}]
   [-DIsplay {FLat_schematic | HIEarchical_schematic}]}
```

Description

Displays single or multiple objects or paths in either schematic window. The objects can be highlighted if desired.

Arguments

- *object_spec*

The value of *object_spec* can be a Tcl list or Tcl collection of one or more names of pin, port, pseudo_port or gate_pin objects.

If *object_spec* is a single object, the object is displayed in the schematic along with its associated instance if it is not already displayed. An error message is generated if the specified object does not exist, or is not of type pin, port, pseudo_port or gate_pin.

The *object_spec* list or collection may also contain lists of names of pins, ports or pseudo ports that form a contiguous path in the schematic in the direction specified by the direction switch. If the path is to be displayed in the hierarchical schematic window, every element that does not have its hierarchical counterpart is dropped from the list. When processing the path, each pair is handled as a segment of the path to be displayed, where one of the objects is a fan-in of the other in the direction specified by the direction switch. All pairs should have the same direction of fan-in and fan-out, otherwise an error is generated. An error is also generated if an object does not exist or there is no valid path between the pair.

- **-direction** {forward | backward}

A required switch and literal pair that specifies the direction of the path provided in *object_spec*.

- **-Highlight** [{0|1|2|3|4|5|none | green | blue | orange | yellow | red }]

An optional switch and optional literal that specifies the highlight color to mark the object or path in the DFTVisualizer schematic window. By default, no highlight is used for marking. Color index values and color mappings are shown in the table below:

Color Index	Color Name
0 (default)	none (default)

Color Index	Color Name
1	green
2	blue
3	orange
4	yellow
5	red

- **-D**isplay {FLat_schematic | HIEarchical_schematic}

An optional switch and literal pair that specifies the schematic window to which the tool should display the specified object or path. By default, the object or path will be displayed in the Flat Schematic window if the -display argument is not provided.

Examples

Example 1

This example shows how to add an object with an instance name of ix249 to the Hierarchical Schematic window, and highlight input pin A red:

```
add_display_path ix249/A -direction forward -highlight red -display hierarchical_schematic
```

Example 2

This example shows how to add instances and display a path between three instances in the Flat Schematic window and highlight the paths yellow:

```
add_display_path {reg_d_1_0/Q ix249/A ix249/Z reg_d_1_1/D} \
    -direction forward -highlight yellow
```

add_drcViolation

Context: dft, patterns

Mode: analysis

Adds a DRC violation to the list of DRC violations for a given DRC rule.

Usage

```
add_drcViolation itcl_object_instance_name [-of_type drc_name]
```

Description

This command is used to create a new DRC violation. The main argument of the command, *itcl_object_instance_name*, is the name of an object of an ITCL class containing a set of prescribed methods. The prescribed and required list of methods are display, report, get_location, and set_simulation_context. The Tessent Shell tool comes with a built-in DrcViolation ITCL class that provides these methods. The body of the class and the method are provided below. The likely hood that you ever need to write your own class is very low as this class is very flexible and complete. For example, if you wanted your display method to create a configuration tree instead of displaying logic in the schematic, then you would derive a new class from the DrcViolation and replace the “display” method with the method you create.

The -of_type argument is typically not used either as it defaults to the *drc_name* which invokes the proc that calls the add_drcViolation command. As the add_drcViolation command is normally used in a proc referred by the “[register_drc -drc_proc](#)” command, it will automatically inherit its *drc_name* from the caller.

You import the ITCL class into your namespace using the following command:

```
namespace import ::mentor::DrcUtilities::*
```

If you wanted to derive a new class to replace the body of a portion of the method, you would use the following:

```
itcl::class MyDrcViolation {
    inherit DrcViolation

    #Replaced methods
    ...
}
```

Refer to examples in the [register_drc](#) command description section for multiple example usages of the class. As shown in those examples, you use this to create an object of the class as follows:

```
set violation [DrcViolation #auto]
```

You then use the dict_set method to populate the information as shown in the code block as follows. If you want to show two paths at the same time, you can also populate the side1 data. The \$path_to_tagged_objects is a list of collections of paths. The paths are typically the ones

created by the “[trace flat model](#) -store_paths_to_tagged_objects” switch. The allowed values for the path_direction are forward and backward. If the \$path_to_tagged_objects contains just one element, you do not need to specify the direction of the path. The \$msg_list variable contains a Tcl list of strings which represent the message that will be displayed when the [analyze_drcViolation](#) or the [report_drcRules](#) commands are run. A new line will be inserted between each element of the message_string_list:

```
$violation dict_set list_of_paths side0 [list $path_to_tagged_objects]
$violation dict_set path_direction side0 backward
$violation dict_set start_callout_string side0 $start_callout_string
$violation dict_set end_callout_string side0 $end_callout_string
$violation dict_set message_string_list $msg_list
```

Once you have finished configuring the ITCL object, you add it as a DRC violation using the following command:

```
add_drcViolation [namespace current] ::$violation
```

Below is the source code defining the DrcViolation as it existed when this documentation was written. You can always see the current list of methods or the implementation of each of the methods by issuing the following commands.

```
info class methods ::mentor::DrcUtilities::DrcViolation
info class definition ::mentor::DrcUtilities::DrcViolation display
```

Notice the use of the environment variable “DrcViolation.echo_commands” and “DrcViolation.no_run_commands” in the display method. If you set the environment variable “DrcViolation.echo_commands” to 1 using the setenv command in your command shell or within the Tessent Tcl shell, the [add_displayPath](#), [add_displayCallout](#), and all other commands used to display the violation will be echoed in the transcript. If you set the environment variable “DrcViolation.no_run_commands” to 1, the command will not run. This is useful to create a regression suite where you can “goldenize” the echoed display commands in the transcript and make sure your custom DRC continues to display the right information without actually having to run the GUI and look at the result.

```

##### DrcViolation.tcl #####
itcl::class DrcViolation {
    private {
        variable dictionary {
            list_of_paths {
                side0 {}
                side1 {}
            }
            path_direction {
                side0 {backward}
                side1 {backward}
            }
            message_string_list {}
            start_callout_string {
                side0 {}
                side1 {}
            }
            end_callout_string {
                side0 {}
                side1 {}
            }
            simulation_context {}
        }
    }
    #
    # Interface
    #
    method dict_set {args} {}
    method dict_get {args} {}
    method report {} {}
    method set_simulation_context {} {}
    method display {{append 0}} {}
    method get_location {} {}
    private method process_cmd {cmd} {}
    private method map_to_design_module_boundary {path_list position} {}
}
#
# Implementation
#
itcl::body DrcViolation::dict_set {args} {
    if {[llength $args] < 2} {
        return_error "Method 'dict_set' of 'DrcViolation' class requires at
least one key and one value."
    } elseif {[lindex $args 0] ni [dict keys $dictionary]} {
        return_error "Method 'dict_set' of 'DrcViolation' class support only
those keys : [join [dict keys $dictionary] ,]."
    }
    if {[lindex $args 0] eq "list_of_paths"} {
        if {[lindex $args 1] ni {side0 side1}} {
            return_error "Key following list_of_paths must be side0 or side1,
not '[lindex $args 1]'"
        }
        set list_of_paths [lindex $args 2]
    }
}

```

```

foreach path $list_of_paths {
    if {[get_attribute_value_list [index_collection $path 0] \
                                -name object_type] eq "gate_pin"} {
        lappend modified_list_of_paths [get_design_objects \
                                         -from_objects $path]
    } else {
        lappend modified_list_of_paths $path
    }
}
dict set dictionary list_of_paths [lindex $args 1] \
                                  $modified_list_of_paths
} elseif {[lindex $args 0] eq "path_direction"} {
    if {[lindex $args 1] ni {side0 side1}} {
        return_error "Key following path_direction must be side0 or side1,
not '[lindex $args 1]'"
    }
    if {[lindex $args 2] ni {backward forward}} {
        return_error "Key following path_direction [lindex $args 1] must be
backward or forward, not '[lindex $args 2]'"
    }
    dict set dictionary {*}$args
} else {
    dict set dictionary {*}$args
}
#####
itcl::body DrcViolation::dict_get {args} {
    if {[llength $args] == 0} {
        return_error "Method 'get' of 'DrcViolation' class requires at least
one key."
    } elseif {[lindex $args 0] ni [dict keys $dictionary]} {
        return_error "Method 'get' of 'DrcViolation' class support only those
keys : [join [dict keys $dictionary] ,]."
    }
    dict get $dictionary {*}$args
}
#####
itcl::body DrcViolation::report {} {
    return [dict get $dictionary message_string_list]
}
#####
itcl::body DrcViolation::set_simulation_context {} {
    if {[dict get $dictionary simulation_context] ne ""} {
        process_cmd "set_current_simulation_context \
                     [dict get $dictionary simulation_context]"
        process_cmd "set_gate_report simulation_context"
    } else {
        process_cmd "set_gate_report normal"
    }
}
#####
itcl::body DrcViolation::display {{append 0}} {
    set echo_commands [expr {
        [info exists ::env(DrcViolation.echo_commands)] && \
        $::env(DrcViolation.echo_commands) eq "1"]
    }
    set no_run_commands [expr {
        [info exists ::env(DrcViolation.no_run_commands)] && \
        $::env(DrcViolation.no_run_commands) eq "1"]
    }]
}

```

```

process_cmd "open_visualizer -display hierarchical_schematic"
if {!$append} {
    process_cmd "delete_display_instances -all \
                 -display hierarchical_schematic"
}
set_simulation_context
foreach side {side0 side1} {
    foreach path [dict get $dictionary list_of_paths $side] {
        if {[sizeof_collection $path] == 0} {
            continue
        }
        set cmd "add_display_path -highlight red \
                  -display hierarchical_schematic \
                  -direction [dict get $dictionary path_direction $side]"
        if {$echo_commands} {
            set display_cmd $cmd
            append display_cmd \
                "\{\n \{[join [get_name_list $path] "\} \\n \{"\} \\n\}"
            puts $display_cmd
        }
        if {!$no_run_commands} {
            append cmd $path
            eval $cmd
        }
    }
    foreach point {start end} {
        set object [map_to_design_module_boundary \
                    [dict get $dictionary list_of_paths $side] $point]
        if {[dict get $dictionary ${point}_callout_string $side] ne "" && \
            ![info exists done(${point}_callout_string,$side)] && \
            [sizeof_collection $object] > 0} {
            if {$echo_commands} {
                set callout_string \
                    [dict get $dictionary ${point}_callout_string $side]
                puts "add_display_callout \"[get_single_name $object ]\" \
                     -string \"$callout_string\" \
                     -display hierarchical_schematic"
            }
            if {!$no_run_commands} {
                eval "add_display_callout $object -string \"$callout_string\" \
                     -display hierarchical_schematic"
            }
        }
    }
}
#####
itcl::body DrcViolation::get_location {} {
    # Location is the start of the path
    if {[llength [dict get $dictionary list_of_paths side0]] > 0 && \
        [sizeof_collection [lindex [dict get $dictionary list_of_paths \
        side0] 0]] > 0} {
        set loc [index_collection [lindex [dict get $dictionary \
        list_of_paths side0] 0] 0]
    } else {
        set loc [index_collection [lindex [dict get $dictionary \
        list_of_paths side1] 0] 0]
    }
}

```

```

if {[get_attribute_value_list $loc -name object_type] eq \
    "pseudo_port"} {
    set loc [get_design_objects -from_objects $loc -map_pseudo_ports]
}
if {[get_attribute_value_list $loc -name object_type] eq "net"} {
    set loc [get_instances -of_nets $loc]
}
if {[sizeof_collection $loc] == 0} {
    set loc [get_current_design]
}
return $loc
}
#####
itcl::body DrcViolation::process_cmd {cmd} {
    upvar no_run_commands no_run_commands
    upvar echo_commands echo_commands
    if {$echo_commands} {
        puts $cmd
    }
    if {!$no_run_commands} {
        eval $cmd
    }
}
#####
itcl::body DrcViolation::map_to_design_module_boundary \
    {path_list position} {
    if {$position ni {start end}} {
        return_error "position is not start or end"
    }
    if {$position eq "end"} {
        for {set i [expr [llength $path_list] -1]} {$i >= 0} {incr i -1} {
            set path [lindex $path_list $i]
            if {[sizeof_collection $path] > 0} {
                return [index_collection $path \
                    [expr [sizeof_collection $path] -1]]
            }
        }
    } else {
        for {set i 0} {$i < [llength $path_list]} {incr i} {
            set path [lindex $path_list $i]
            if {[sizeof_collection $path] > 0} {
                return [index_collection $path 0]
            }
        }
    }
}
}

```

Arguments

- *itcl_object_instance_name*

A required string containing the name of an object of an iTcl class. As described above, the iTcl class must have a prescribed set of required methods. Refer to the description above for a description of those methods and the description of the built-in DrcViolation class that you can use for your custom DRC violation.

- **-of_type** *drc_name*

An optional switch-value pair used to specify the DRC name for which the violation belongs. The **-of_type** argument is typically not used either as it defaults to the *drc_name* which invoke the proc that called the **add_drcViolation** command. As the **add_drcViolation** command is normally used in a proc referred by the “[register_drc](#) -**drc_proc**” switch, it will automatically inherit its *drc_name* from the caller.

Examples

The following example usage of the **add_drcViolation** command is taken from one of the many examples shown in the [register_drc](#) command examples. The built-in **DrcViolation** class is used to create the **\$drcViolation** object. The **dict_set** method is used to configure its data. After that the **add_drcViolation** command is issued to create the violation.

```

if {[sizeof_collection $multi_input_gates] > 0} {
    append_to_collection multi_input_cells \
        [get_instances -of_gate_pins $multi_input_gates -silent] -unique
    set multi_input_cell_count [sizeof_collection $multi_input_cells]
    set violation [DrcViolation #auto]
    $violation dict_set list_of_paths side0 \
        [list $path_to_tagged_objects]
    $violation dict_set path_direction side0 backward
    $violation dict_set start_callout_string side0 \
        "PO with path to PI with multi input gates along the path"
    $violation dict_set end_callout_string side0 "PI in fanin of PO"
    set start_port [index_collection $path_to_tagged_objects 0]
    set end_port [index_collection $path_to_tagged_objects \
        [expr [sizeof_collection $path_to_tagged_objects] - 1]]
    lappend msg_list "The output port '[get_single_name $start_port]' has a
combinational path from the input port '[get_single_name $end_port]'"
    lappend msg_list "but the path is not a pure feed-through as the path
crosses $multi_input_cell_count multi-input cells:"
    set cnt 1
    set max 3
    foreach_in_collection multi_input_cell $multi_input_cells {
        if {$cnt > $max} {
            lappend msg_list "... "
            break
        }
        lappend msg_list "  [get_single_name $multi_input_cell]"
        incr cnt
    }
    $violation dict_set message_string_list $msg_list
    add_drcViolation [namespace current]::$violation
}

```

add_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.

Usage

`add_edt_blocks block_name`

Description

Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.

This command defines a string, referred to as the block name or block tag, as the identifier for an EDT block instantiated in a design netlist and simultaneously restricts the applicability of certain subsequent commands to that block. Restricting the applicability of these commands to a particular EDT block is referred to as setting or changing the EDT context, and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the [delete_edt_blocks](#) command or change the context to a different EDT block using [set_current_edt_block](#) or another add_edt_blocks command.

You must use this command to define a unique name for an EDT block prior to defining its attributes using context-sensitive commands. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (set_system_mode, create_patterns and report_statistics for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block (“set_edt_options -channels” for example) returns an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- ***block_name***

A required string that specifies a name to use for uniquely identifying a specific EDT block instantiated within a netlist. The name of the block must begin with an alphabetic character and contain only alphabetic characters, numeric characters, and the underscore (a-zA-Z0-9_). Consecutive underscores are not allowed. Underscores cannot appear as the

add_edt_blocks

first or last character of the identifier. The block_name is independent of the module and instance names used in the netlist. Escaped identifiers are not allowed.

Examples**Example 1**

The following example defines the block name, “my_core1_block” for an EDT block that contains four scan chains. The tool determines the EDT block to associate with this name from the pin pathnames specified in the [add_scan_chains](#) commands that follow the add_edt_blocks command.

```
add_edt_blocks my_core1_block
# my_core1_block is the current EDT block.

...
add_scan_chains -internal core1_chain1 grp1 /edt_top_core1/cpu1_i/edt_si1 \
/edt_top_core1/cpu1_i/edt_so1
add_scan_chains -internal core1_chain2 grp1 /edt_top_core1/cpu1_i/edt_si2 \
/edt_top_core1/cpu1_i/edt_so2
add_scan_chains -internal core1_chain3 grp1 /edt_top_core1/cpu1_i/edt_si3 \
/edt_top_core1/cpu1_i/edt_so3
add_scan_chains -internal core1_chain4 grp1 /edt_top_core1/cpu1_i/edt_si4 \
/edt_top_core1/cpu1_i/edt_so4
```

Example 2

The following example defines two more block names, “my_core2_block” and “my_core3_block”, then displays a list of the current block names. The display indicates subsequent context-sensitive commands will apply only to the EDT block named “my_core3_block”.

```
add_edt_blocks my_core2_block
# my_core2_block is the current EDT block.

...
add_scan_chains -internal core2_chain1 grp1 /edt_top_core2/cpu3_i/edt_si1 \
/edt_top_core2/cpu3_i/edt_so1
add_scan_chains -internal core2_chain2 grp1 /edt_top_core2/cpu3_i/edt_si2 \
/edt_top_core2/cpu3_i/edt_so2
...

add_edt_blocks my_core3_block
# my_core3_block is the current EDT block.

...
add_scan_chains -internal core3_chain1 grp1 /edt_top_core3/cpu7_i/edt_si1 \
/edt_top_core3/cpu7_i/edt_so1
add_scan_chains -internal core3_chain8 grp1 /edt_top_core3/cpu7_i/edt_si8 \
/edt_top_core3/cpu7_i/edt_so8

report_edt_blocks
```

```
// my_core1_block  
// my_core2_block  
// my_core3_block      (current block)
```

Example 3

The following example removes from the tool's internal database, the data for the EDT block named "my_core3_block", making it unavailable for use with the [set_current_edt_block](#) command.

```
delete_edt_blocks my_core3_block  
// Block deleted. No EDT block now selected as current block.  
report_edt_blocks  
// my_core1_block  
// my_core2_block
```

Related Topics

[delete_edt_blocks](#)
[report_edt_blocks](#)
[set_current_edt_block](#)

add_edt_configurations

Context: dft -edt

Mode: setup

Sets up EDT logic with two compression configurations.

Usage

`add_edt_configurations config_name`

Description

Sets up EDT logic with two compression configurations.

The `add_edt_configurations` command creates a compression configuration and applies subsequent EDT channel specifications to the configuration. Up to two compression configurations can be defined for a design block.

In a multi-block design flow, you can define the compression configurations at the top-level integration and all subsequently added blocks inherit the defined compression configurations. The configuration parameters need to be set for each block context.

Note

 When two configurations are specified and one of the configurations is missing an input/output channel parameter, it inherits the needed value from the other configuration. You can override this feature by explicitly specifying the channel configuration.

For more information on setting up two compression configurations, see the “[Defining Dual Compression Configurations](#)” section in the *Tessent TestKompress User’s Manual*.

Arguments

- `config_name`

A required string that specifies a name for the compression configuration.

Examples

The following example shows a dofile that creates two configurations for a single block.

```
# edt_ip_creation.do
#
# Dofile for EDT logic creation
dofile scan_chain_setup.dofile

# Set up EDT configurations
set_edt_options -location internal
add_edt_configurations my_sys_test_config
set_edt_options -input_channels 4 -output_channels 2
add_edt_configurations my_wafer_test_config
set_edt_options -input_channels 5 -output_channels 4

# Set bypass pin
set_edt_pins bypass my_bypass_pin

# Set edt configuration pin
set_edt_pins configuration my_configuration_pin
set_system_mode analysis

# Report and write EDT logic.
report_edt_configurations -all # Report configurations for all blocks.
report_edt_pins # Report all pins including compression configuration
# specific pins.
write_edt_files created -verilog -replace # Create dofiles and
# test procedure files for both the configurations and the bypass mode

write_design -output_file created_edt_top_rtl.v -replace # Write the
# current design in Verilog format to the file created_edt_top_rtl.v.
```

Related Topics

[delete_edt_configurations](#)
[report_edt_configurations](#)
[set_current_edt_configuration](#)

add_false_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.

Usage

```
add_false_paths
{[-From source_node... | -FROM_Clock clock_signal_name...]
 [-TO sink_node... | -TO_Clock clock_signal_name...]
 [{-THrough through_node...}...]} | -CROSS_CLOCK_DOMAINS
 [-Hold] [-Setup] [-MAX_Skew {HALf_cycle | ONE_cycle | MULtiple_cycles}]
```

Description

Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.

If a false path is sensitized for an at-speed test during ATPG, incorrect capture responses can occur at certain scan cells and POs with resulting incorrect capture responses at the end points of the false path. This can cause simulation mismatches when you verify the patterns in a timing-based simulator. By defining the false paths, you enable the tool to identify the incorrect capture responses. Once identified, the tool will mask these responses (modify them to X) in the resultant patterns so they will not be the source of simulation mismatches.

For additional information, refer to “[Pattern Failures Due to Timing Exception Paths](#),” in the *Tessent Scan and ATPG User’s Manual*.

You can use the “[set_timing_exceptions_handling -allow_invalid_pin_names on](#)” command to allow the add_false_paths command to issue warnings instead of errors when defined objects are not found.

Arguments

Tip

i When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -From *source_node*...

An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins

of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

By default, if a specified source node does not translate to an internal gate, the tool will transcript an error message, and not add the false path definition to its internal database. If you issue the “[set_timing_exceptions_handling -allow_invalid_pin_name on](#)” command, the tool will issue a warning and add the false path definition.

- **-FROM_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

By default, if a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database. If you issue the “[set_timing_exceptions_handling -allow_invalid_pin_name on](#)” command, the tool will issue a warning and add the false path definition.

- **-TO *sink_node*...**

An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

By default, if a specified sink node does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database. If you issue the “[set_timing_exceptions_handling -allow_invalid_pin_name on](#)” command, the tool will issue a warning and add the false path definition.

Note

 For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “[add_false_paths... -to CLK2](#)” would define paths with an end point at D2.

- **-TO_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with

this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

By default, if a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database. If you issue the “[set_timing_exceptions_handling -allow_invalid_pin_name on](#)” command, the tool will issue a warning and add the false path definition.

- {-THrough *through_node...*}...

An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

```
add_false_paths -from A -through B C -through D E -to F
```

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool will consider all paths from the specified starting points to the specified end points to be false paths.

By default, if a through node does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database. If you issue the “[set_timing_exceptions_handling -allow_invalid_pin_name on](#)” command, the tool will issue a warning and add the false path definition.

- **-Cross_clock_domains**

A switch that adds hold time false paths between all clocks.

Note

 You cannot use this switch in Setup mode.

Use this switch instead of multiple `add_false_paths` command instances using the `-TO_Clock` and `-FROM_Clock` switches to specify each path and direction between the clocks. See Example 7.

- **-Hold**

An optional switch that evaluates the false path for hold time exceptions.

By default, the tool considers the hold time false path only when the `-from` flop and the `-to` flop are updated at the same clock edge for most paths. This behavior matches the “same edge” interaction of the domain clock setting. The exception is when you use the “`add_false_paths -cross_clock_domains`” or the false paths derived from asynchronous clock groups defined in SDC files. For such cross clock domain false paths, the tool assumes that the clock skew can be multiple cycles.

When you define a false path without the -hold and the -setup options, the default is to consider the false path with both the hold and setup time exceptions.

- **-Setup**

An optional switch that evaluates the false path for setup time exceptions.

For more information, see “[Types of Timing Exception Paths](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-MAX_Skew**

An optional switch that specifies the maximum clock skew for the hold time exception for this false path. This option overrides the global -max_skew specified by the set_timing_exceptions_handling command.

[HALf_cycle](#)

Specifies half cycle as the maximum clock skew. This is the default value for this option, unless -cross_clock_domains is specified, in which case, the max skew is multiple cycle.

[ONE_cycle](#)

Specifies one cycle as the maximum clock skew.

[MULtiple_cycles](#)

Specifies multiple cycles as the maximum clock skew.

Examples

Example 1

The following example creates three false paths definitions, then reports all current false path definitions:

```
add_false_paths -from /my_design/a /my_design/b -through /my_design/u1
add_false_paths -to /my_design/c
add_false_paths -through /my_design/u24 /my_design/u25
report_false_paths -all

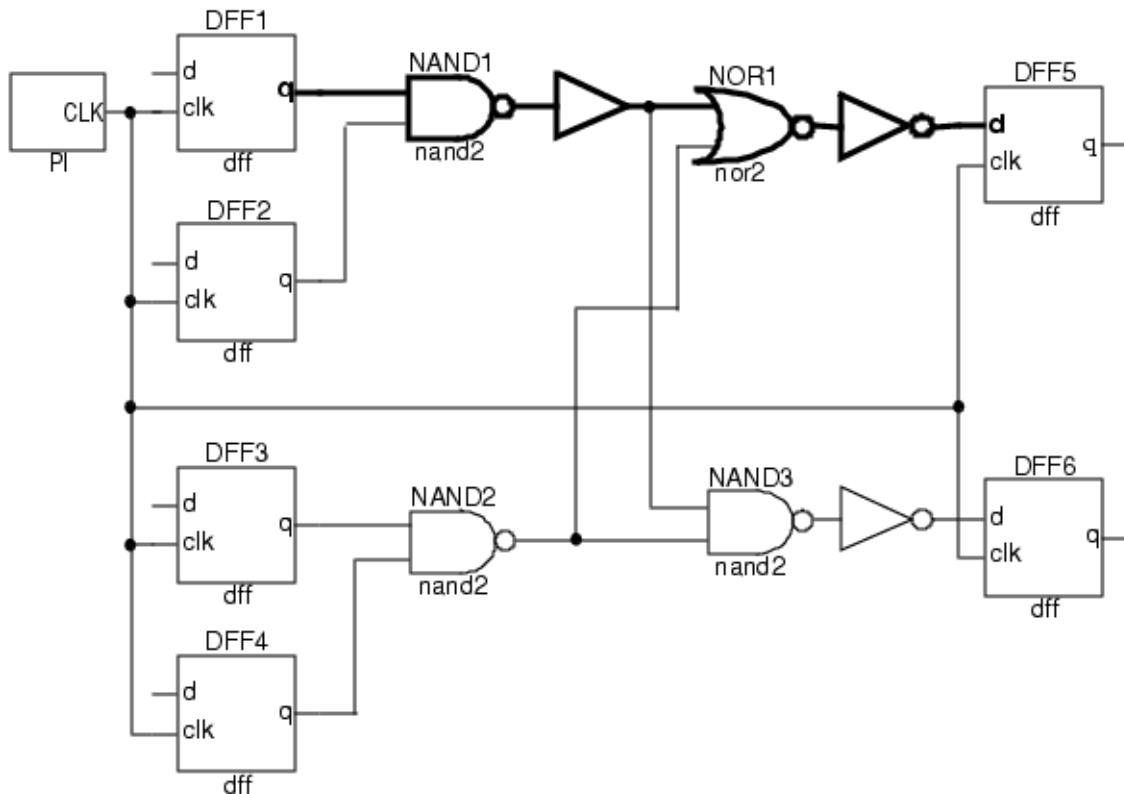
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3
```

Example 2

The following example defines the path highlighted in [Figure 3-12](#) as a false path:

```
add_false_paths -from DFF1/q -to DFF5/d
```

Figure 3-12. Example of a False Path to be Defined



Here are four more ways you could define the same false path:

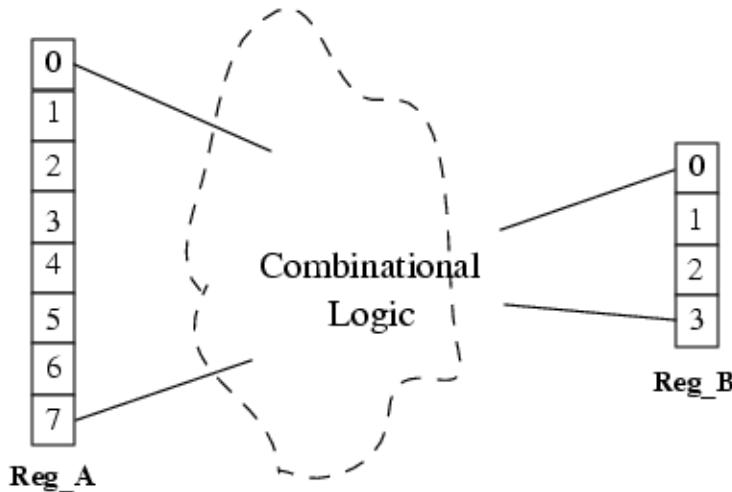
```
add_false_paths -from DFF1/q -through NOR1 -to_clock CLK  
add_false_paths -from DFF1/clk -to DFF5/d  
add_false_paths -from DFF1/q -to DFF5/clk  
add_false_paths -from DFF1/clk -to DFF5/clk
```

Example 3

Referring to [Figure 3-13](#), the following example defines as false paths all the paths that start from Reg_A[0...7] and end at Reg_B[0...3]:

```
add_false_paths -from Reg_A -to Reg_B
```

Figure 3-13. Example Circuit for Defining False Paths



Example 4

This example defines as false paths (in [Figure 3-13](#)) all the paths that start from Reg_A[0] and end at Reg_B[0...3]:

```
add_false_paths -from Reg_A[0] -to Reg_B
```

Example 5

This example defines as false paths all the paths that start from Reg_A[0] or Reg_A[3] and end at Reg_B[2]:

```
add_false_paths -from Reg_A[0] Reg_A[3] -to Reg_B[2]
```

Example 6

The remaining six examples describe signal matches for various sample pin pathnames specified using the asterisk (*) wildcard character. The pathnames are based on the following design:

```
module top(...);
    Input ...;
    Output...;

    ...
    foo u1(...);

    ...
endmodule; module foo(...);
    Input...;
    Output...;

    ...
    staticRegister I_reg_1(.D1(...), .C1(...), .Z(...));
    staticRegister I_reg_2(.D1(...), .C1(...), .Z(...));
    staticRegister I_reg_3(.D1(...), .C1(...), .Z(...));
    staticRegister F_reg_1(.D1(...), .C1(...), .Z(...));
    staticRegister F_reg_2(.D1(...), .C1(...), .Z(...));
    ...
endmodule;
```

add_false_paths

1. The signal name /u1/*reg_1/D1 will map to signals /u1/I_reg_1/D1 and /u1/F_reg_1/D1.
2. The signal name /u1/I_reg*/D1 will map to signals /u1/I_reg_1/D1, /u1/I_reg_2/D1 and /u1/I_reg_3/D1.
3. The signal name /u1/*reg*/Z will map to signals /u1/I_reg_1/Z, /u1/I_reg_2/Z, /u1/I_reg_3/Z, /u1/F_reg_1/Z and /u1/F_reg_2/Z.
4. The signal name /u1/F_reg_2/* will map to signals /u1/F_reg_2/D1, /u1/F_reg_2/C1 and /u1/F_reg_2/Z.
5. The signal name /u1/F*/*1 will map to signals /u1/F_reg_1/D1, /u1/F_reg_1/C1, /u1/F_reg_2/D1 and /u1/F_reg_2/C1.
6. The signal name /u1/I*1/C1 will map to signal /u1/I_reg_1/C1.

Example 7

Assume the design contains two defined clocks: clk1 and clk2. The following command adds hold time false paths between clk1 and clk2:

```
add_false_paths -cross_clock_domains -hold
```

The above command is equivalent to the following two commands:

```
add_false_paths -from_clock clk1 -to_clock clk2 -hold
add_false_paths -from_clock clk2 -to_clock clk1 -hold
```

Related Topics

[delete_false_paths](#)
[delete_multicycle_paths](#)
[read_sdc](#)
[report_false_paths](#)
[report_multicycle_paths](#)

add_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Automatically adds fault site definitions for library cells that do not have UDFM definitions.

Usage

```
add_fault_sites {-ALI | -UNDEFINED_Cells} [-VERBose]
```

Description

Automatically adds fault site definitions for library cells that do not have UDFM definitions.

The add_fault_sites command automatically adds port fault site definitions for library cells that are not covered by the current UDFM definitions so that you do not have to add them manually to the UDFM file. In order to distinguish those added definitions and resulting fault sites in report functions, the added definitions are collected in the UDFM type “auto-inserted-faults.”

After you’ve added the fault sites, the [add_faults](#) command adds static or delay faults to those locations, as specified by the “[set_fault_type udfm](#)” command.

Note

 The add_fault_sites command works only for the fault type UDFM, and is available only after you read a UDFM fault file and before you create the fault list.

For more information about UDFM, refer to “[About User-Defined Fault Modeling](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **-ALI**

A required switch that automatically adds UDFM definitions for all undefined (uncovered) port fault sites, including PI’s and PO’s. The fault sites affected by this switch include all those affected by the -Undefined_Cells switch plus remaining port fault site definitions for existing cells and modules, as well as additional port fault sites without this scope.

- **-UNDEFINED_Cells**

A required switch that automatically adds UDFM definitions for cell port fault sites in undefined (uncovered) cells or modules.

- **-VERBose**

An optional switch that displays additional information about each added UDFM definition.

Examples

The following example shows the effect of adding fault site definitions for all library cells not covered by UDFM definitions:

```
read_fault_sites /home/design/c090_std/c090_std.udfm
//  Reading UDFM File: /home/design/c090_std/c090_std.udfm
//  UDFM definitions read: Total=3, Cells=1, Modules=2, Instances=0,
//  No Match=0

report_fault_sites -undefined_cells
//  Report of cells not defined by UDFM
//
//  Cell Name  Instances
//  -----  -----
//  M1          2
//  M2          1
//  u_mux2      1
//  -----  -----


add_fault_sites -undefined_cells
//  3 UDFM definition(s) for uncovered cells have been added

add_faults -all
//  Number added faults=56  instances considered=7  not-considered=3
```

Related Topics

[add_faults](#)
[delete_faults](#)
[read_fault_sites](#)
[report_fault_sites](#)
[set_fault_type](#)
[write_fault_sites](#)

add_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected (actual category is UC).

Usage

add_faults *fault_model_specific_options* [*power_aware_options*] [-KEEP_Patterns]

Path Delay Faults Usage:

```
add_faults {{name_of_delay_path}... | -All} [-Both | -Rise | -Fall] [-VERbose]
[>|>>} file.pathname]
```

Stuck/Transition/Toggle/Iddq Faults Usage:

```
add_faults {{ -All | {object_expression...
[-PIN | -INstance | -MODule [-PIN module_pin_pathname ...]]}} | 
{-CLOCK_Domains {ALL | clock_pathname...} [-NO_EQUIvalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]} | 
{-CAPture_procedures {ALL | capture_procedure_name...}}}
[-Stuck_at {01 | 0 | 1}] [-VERbose] [>|>>} file.pathname]
```

Bridge Faults Usage:

```
add_faults {{-All | {bridge_name... -NAME} | {net_pathname... -SINGle} |
instance_name... -SINGle}} | 
{-CLOCK_Domains {ALL | clock_pathname...} [-NO_EQUIvalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]} | 
{-CAPture_procedures {ALL | capture_procedure_name...}} [>|>>} file.pathname]
```

UDFM Faults Usage:

```
add_faults {-All | {object_expression... [-INstance]}}
| {-CLOCK_Domains {ALL | clock_pathname...} [-NO_EQUIvalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}
{[-UDFM_type name][-CELLname] [-MODule name] [-FAULT name] } [-VERbose]
```

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):

```
[[ -ON_domains] | [-OFf_domains] | [-POWer_domains {domain_name ...}]]
[-ISolation_cells] [-LLevel_shifters] [-REtention_cells]
```

Description

Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected (actual category is UC).

When you enter the Setup mode, all faults from the current fault list are deleted. Furthermore, if you change the fault type, all faults are deleted. The power-aware options add faults based on power domains and/or power features after you have loaded a CPF or UPF file.

You cannot add faults for clock domains, capture procedures, and specific objects or paths within the same command instance. Use a separate command instance for each.

With the fault mode set to “uncollapsed” (the invocation default), all possible faults are added to the list. If you change the fault mode to “collapsed” with the `set_fault_mode` command, only one instance of any given fault is added, ignoring any equivalent faults.

Arguments

- **-KEEP_Patterns**

An optional switch that preserves the current test pattern set and maintains the status of the current faults while adding faults. The default behavior is to discard the current test pattern set and set all faults to undetected.

- **-All**

A required switch that adds all faults, depending on the fault model.

- **Bridge Faults** — Adds all previously loaded bridge entries to the internal fault list. This is the default behavior.
- **UDFM Faults** — Adds faults for all previously loaded UDFM fault sites to the internal fault list. This is the default behavior.
- **Non-Bridge Faults** — Adds all faults on all model, netlist primitive, and top module pins. The setting of the `set_internal_fault` command affects the behavior of this switch. If the `set_internal_fault` command is set to “off” (the default), the switch only adds faults residing on the boundaries of library cells. If the `set_internal_fault` command is set to “on”, the `-All` switch adds faults that reside on the inputs and outputs of gates within library cells. For more information about fault locations, refer to “[Fault Locations](#)” in the Tesson Scan and ATPG User’s Manual.

- ***name_of_delay_path***

A required, repeatable string that specifies the names of the delay paths whose faults you want added to the current fault list. The name must correspond to the name of a delay path defined in [the path definition file](#).

- **-Both | -Rise | -Fall**

An optional switch that specifies which faults to add for each path already added via the `read_fault_sites` command. These switches are used for path delay faults only.

- **-Both** — Adds both the slow-to-rise and slow-to-fall faults. This is the default.
- **-Rise** — Adds only the slow-to-rise faults.
- **-Fall** — Adds only the slow-to-fall faults.

- ***object_expression***

A required, repeatable string that specifies a list of instances or pins whose faults you want added to the current fault list. Note that in the case of UDFM, you can specify only

instances. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

Instance pathnames must be Tessent Cell library cell instances. Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- **-PIN**

An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then add faults for all the pins matched.

- **-INstance**

An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then add faults for all the pins on the instances matched.

- **-MODule [-PIN *module_pin_pathname* ...]** (Stuck/Transition/Toggle/Iddq only)

A switch that specifies to interpret the *object_expression* argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the *object_expression* argument, and the tool adds the fault for all matching modules or library models.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and adds faults for every pin specified in module_pin_pathname list.

- ***bridge_name***

A required, repeatable string that specifies bridge names whose bridge faults you want added to the current fault list.

- ***net_pathname***

A required, repeatable string that specifies single net names whose bridge faults you want added to the current fault list.

- ***instance_name***

A required, repeatable string that specifies instances names whose bridge faults you want added to the current fault list.

- **-NAME**

An optional switch that specifies all entered strings are bridge names.

- **-SINGle**

An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

If the string specifies a net, faults are added to the fault list for each bridge entry associated with the net.

If the string specifies an instance that contains a non-dominant net specified in a bridge entry, associated faults are added to the fault list.

- **-CLOCK_Domains {ALL | *clock.pathname...*}**

A required switch and literal or repeatable string pair that specifies a list of clocks which is used by the tool to decide the faults to be added to the fault list, given the following requirements are met:

- **Static Faults** — A fault is added to the fault list if it can be captured by any clock in the specified list of clocks or any of its equivalent clocks.
- **Transition Faults** — A fault is added to the fault list:
 - i. If the launch and capture clock are the same clock from the specified list of clocks
 - ii. If the launch and capture clock are synchronous clocks from the specified list of clocks
 - iii. If the launch or capture clock from the specified are equivalent equivalent clocks

When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

The argument choices are as follows:

ALL — A literal that specifies all the clocks in the design.

clock.pathname — A repeatable string that specifies individual clocks.

Note

 A fault added using this switch might also be detectable by an unspecified clock.

- **-NO_EQUIvalent_clocks**

An optional switch that prevents the -Clock_domains switch from adding faults in equivalent clock domains.

- **-EXCLUDEFAULTSBETWEENSYNCclockdomains**

An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When specified, only faults within clock domains are considered when adding or deleting faults by clock domain.

- **-CAPture_procedures{ALL | *capture_procedure_name...*}**

A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to add faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

ALL — A literal that specifies all enabled named capture procedures.

capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

For information about enabling or disabling named capture procedures, see the [set_capture_procedures](#) command description.

- **-Stuck_at {01 | 0 | 1}**

An optional switch and literal pair that specifies which stuck-at or transition faults to add to the fault list.

01 — A literal specifying that, for stuck-at faults, the tool add both the “stuck-at-0” and “stuck-at-1” faults; or for transition faults, the tool add both “slow-to-rise” and “slow-to-fall” faults. This is the default.

0 — A literal specifying that, for stuck-at faults, the tool add only the “stuck-at-0” faults; or for transition faults, the tool add only “slow-to-rise” faults.

1 — A literal specifying that, for stuck-at faults, the tool add only the “stuck-at-1” faults; or for transition faults, the tool add only “slow-to-fall” faults.

- **-Udfm_type *name***

A switch and string pair that specifies a UDFM type. Use this switch to add faults associated with a specified UDFM type. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-CELL *name***

A switch and string pair that specifies a library cell. Use this switch to add faults associated with a specified library cell. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-MODule *name (UDFM only)***

A switch and string pair that specifies a module. Use this switch to add faults associated with a specified module. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-VERbose**

By default, when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, a summary message similar to the following is output:

```
// Note: Adding faults for 330 fault sites.
```

When you specify the optional -VERbose switch, the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

When you specify the -Udfm switch and include the -Verbose option, the tool outputs a message similar to the following:

```
// Note: Adding 627224 for 27398 instances
```

- >*file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.

- >>*file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

An optional switch that adds faults on all power-on domains.

- -OFF_domains

An optional switch that adds faults on all power-off domains.

- -Power_domains {*domain_name* ...}

An optional switch and repeatable string that adds faults on the specified power domains (*domain_name*).

- -Isolation_cells

An optional switch that adds faults on all isolation cells.

- -Level_shifters

An optional switch that adds faults on all level shifters.

- -REtention_cells

An optional switch that adds faults on all retention cells.

Examples

Example 1

The following example adds all faults to the circuit so you can run an ATPG process:

```
set_system_mode analysis
create_patterns
```

Example 2

The following example uses a prefix match to add faults only to pins on instances whose pathnames begin with “/u9/u2/LCT_reg”:

```
add_faults /u9/u2/LCT_reg*
```

```
==== Found 16 design pins ===
/u9/u2/LCT_reg_0_/A0
/u9/u2/LCT_reg_0_/A1
/u9/u2/LCT_reg_0_/S0
/u9/u2/LCT_reg_0_/Y
/u9/u2/LCT_reg_1_/A0
/u9/u2/LCT_reg_1_/A1
/u9/u2/LCT_reg_1_/S0
/u9/u2/LCT_reg_1_/Y
/u9/u2/LCT_reg_2_/A0
/u9/u2/LCT_reg_2_/A1
/u9/u2/LCT_reg_2_/S0
/u9/u2/LCT_reg_2_/Y
/u9/u2/LCT_reg_3_/A0
/u9/u2/LCT_reg_3_/A1
/u9/u2/LCT_reg_3_/S0
/u9/u2/LCT_reg_3_/Y
```

Example 3

The following example uses the same prefix match as the preceding example, but includes the -Instance switch:

```
delete_faults -all
add_faults /u9/u2/LCT_reg* -instance

==== Found 4 design instances ===
/u9/u2/LCT_reg_0_
/u9/u2/LCT_reg_1_
/u9/u2/LCT_reg_2_
/u9/u2/LCT_reg_3_
```

Notice that when you use the -Instance switch, the command lists the instances whose pins are faulted as a result of the wildcard character rather than listing each individual pin.

Example 4

The following example demonstrates how to wildcard the second level of hierarchy:

```
add_faults /u9/*/LCT_reg_0_ -instance

==== Found 4 design instances ===
/u9/u4/LCT_reg_0_
/u9/u3/LCT_reg_0_
/u9/u2/LCT_reg_0_
/u9/u1/LCT_reg_0_
```

Example 5

The following example combines a wildcard character at the second level of hierarchy with a prefix match:

```
add_faults /u9/*/LCT_reg* -instance
```

```
==== Found 32 design instances ===
/u9/u4/LCT_reg_0_
/u9/u4/LCT_reg_1_
/u9/u4/LCT_reg_2_
/u9/u4/LCT_reg_3_
/u9/u4/LCT_reg_4_
/u9/u4/LCT_reg_5_
/u9/u4/LCT_reg_6_
/u9/u4/LCT_reg_7_
/u9/u3/LCT_reg_0_
/u9/u3/LCT_reg_1_
/u9/u3/LCT_reg_2_
...
/u9/u2/LCT_reg_5_
/u9/u2/LCT_reg_6_
/u9/u2/LCT_reg_7_
/u9/u1/LCT_reg_0_
/u9/u1/LCT_reg_1_
/u9/u1/LCT_reg_2_
/u9/u1/LCT_reg_3_
/u9/u1/LCT_reg_4_
/u9/u1/LCT_reg_5_
/u9/u1/LCT_reg_6_
/u9/u1/LCT_reg_7_
```

Example 6

The following example adds only faults in the power domain named PD1:

add_faults -power_domain PD1

Example 7

The following example uses a prefix match to add faults to pins on instances whose pathnames begin with “/u9/u2/LCT_reg” plus all level shifters in the design:

add_faults /u9/u2/LCT_reg* -level_shifters

Example 8

The following example adds *intra_cell_bridges* UDFM fault models associated with all library cells that start with *mux* to the current fault list.

add_faults -all -udfm_type intra_cell_bridges -cell mux*

Example 9

The following example adds *intra_cell_bridges* UDFM fault models to the instance *my_module1_ins/my_module1_or02*. Note, when adding a UDFM fault model, the *-instance* switch is not used.

add_faults my_module1_ins/my_module1_or02 -udfm_type intra_cell_bridges

Example 10

The following example adds fault models to every module in the current fault list whose name begins with *adder*.

add_faults -all -module adder*

Example 11

The following example displays the fault value on the specified gate_pin after the add_faults command is executed:

```

add_nofaults ha1/ddd/q -Pin -Stuck 1
add_nofaults ha1/ddd/clk -Pin -Stuck 0
add_nofaults ha1/ddd/d -Pin -Stuck 01
....
add_faults -all
....

foreach_in_collection i [get_gate_pins ha1/ddd/* ] {
    puts "[get_name_list$i] fault_site:[get_attribute_value_list$i-name fault_site]"
}

ha1/ddd/clk    fault_site: 1
ha1/ddd/d     fault_site: none
ha1/ddd/q     fault_site: 0

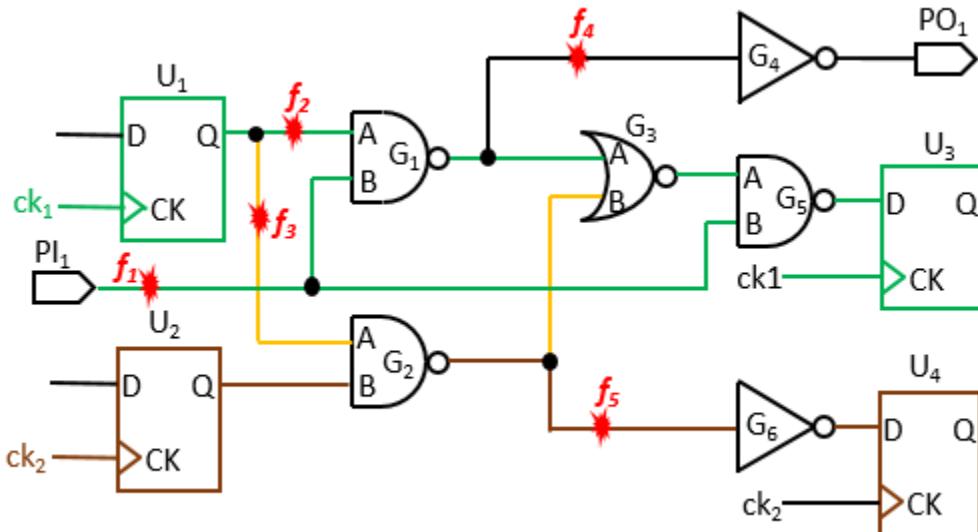
```

Example 12

Using the -clock_domains switch with stuck-at faults:

add_faults -clock_domains ck1 ck2

Figure 3-14. Static Fault



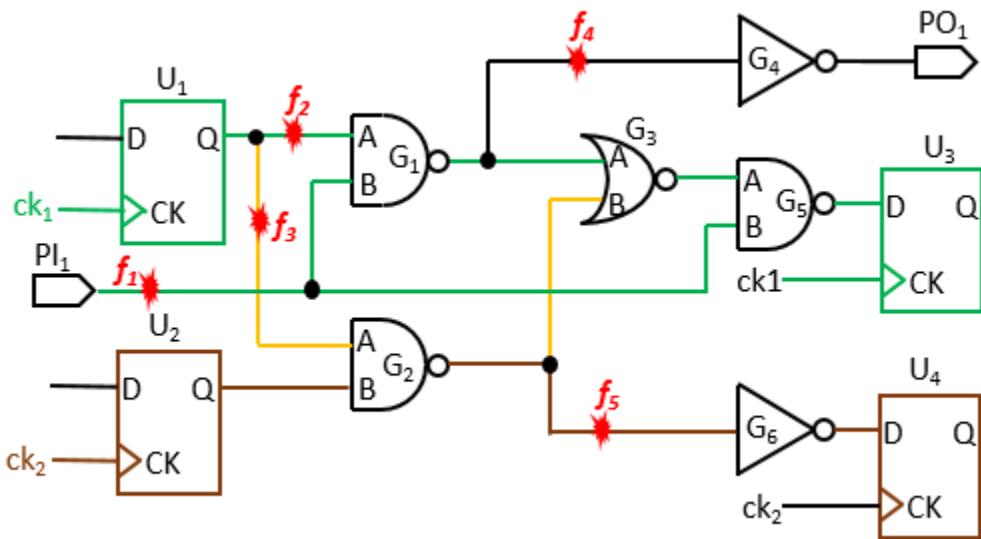
The included faults are f_1 , f_2 , f_3 , and f_5 .

Example 13

For this example, assume that f_1 , f_2 , f_3 , f_4 , and f_5 are legitimate fault sites. Other pins are no-faulted. Using the -clock_domains switch with transition faults, without defining synchronous clock group. In this case, ck_1 and ck_2 are incompatible.

```
add_faults -clock_domains ck1 ck2
```

Figure 3-15. Transition Fault - Incompatible Clocks



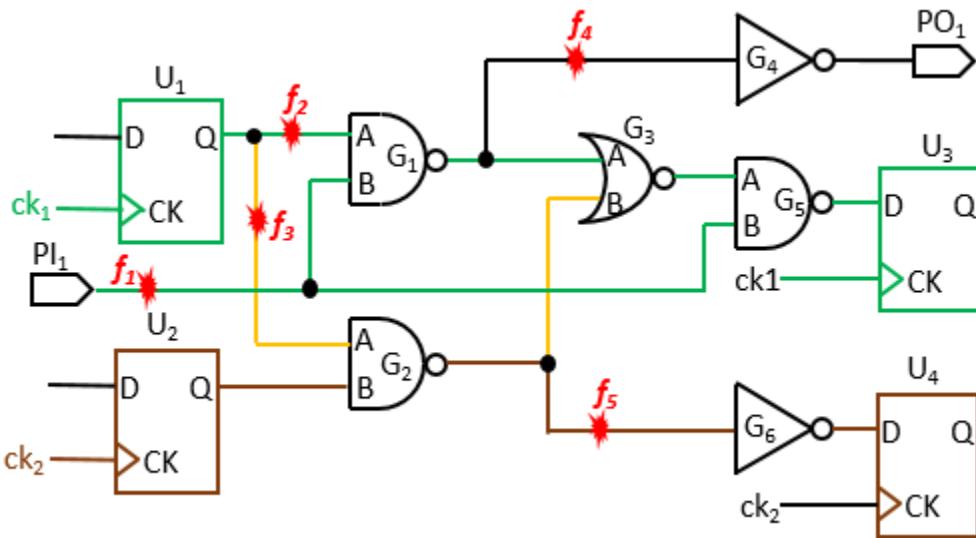
The added faults are f_2 , and f_5 . In this case, f_3 is not added because ck_1 (launch clock) and ck_2 (capture clock) are asynchronous.

Example 14

Using the -clock_domains switch with transition faults, after defining a synchronous clock group. In this case, ck_1 and ck_2 are compatible:

```
add_synchronous_clock_group {ck1 ck2}  
...  
add_faults -clock_domains ck1 ck2
```

Figure 3-16. Transition Fault - Compatible Clocks



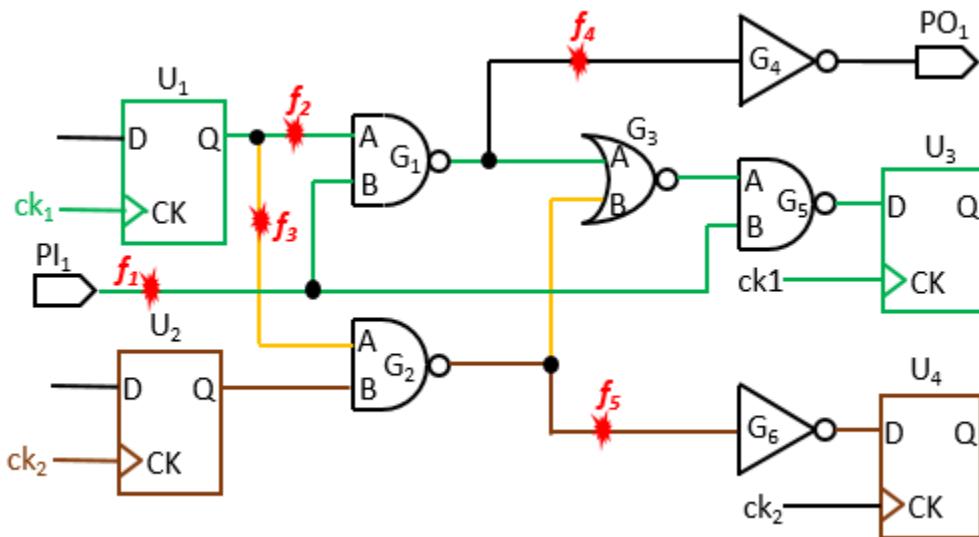
The added faults, launched and captured by ck_1 and ck_2 , are f₂, f₃, and f₅. You can use the `add_faults` option `-exclude_faults_between_synch_clock_domains` to exclude cross domain faults, such as f₃.

Example 15

For this example, assume that f₁, f₂, f₃, f₄, and f₅ are legitimate fault sites. Other pins are no-faulted. Using the `-clock_domains` switch with transition faults, without defining equivalent pins:

```
add_faults -clock_domains ck1
```

Figure 3-17. Transition Fault - No Equivalent Pins



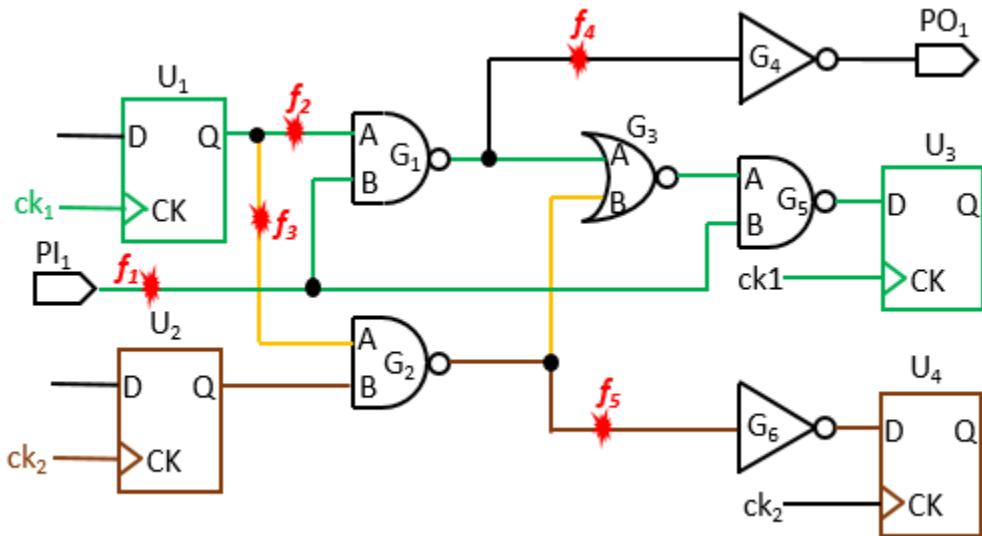
The added fault is f_2 .

Example 16

For this example, assume that f_1 , f_2 , f_3 , f_4 , and f_5 are legitimate fault sites. Other pins are no-faulted. Using the -clock_domains switch with transition faults, with pin equivalents defined for ck_1 and ck_2 :

```
add_input_constraints -equivalent ck1 ck2
...
add_faults -clock_domains ck1
```

Figure 3-18. Transition Fault - Equivalent Pins



The added faults are f_2 , f_3 , and f_5 . You can use the add_faults option -no_equivalent_clocks to prevent the addition of faults in equivalent clock domains. In this case, faults f_3 and f_5 will not be added.

Related Topics

[create_patterns](#)
[delete_faults](#)
[read_fault_sites](#)
[read_faults](#)
[read_cpf](#)
[read_upf](#)
[report_faults](#)
[report_testbench_simulation_options](#)

[set_fault_mode](#)
[set_fault_sampling](#)
[set_fault_type](#)
[set_internal_fault](#)
[write_faults](#)

add_icl_ports

Context: patterns -ijtag, dft

Mode: setup, insertion

Specifies top design ports that will be added in the ICL file generated during ICL extraction.

Usage

```
add_icl_ports port_objects [-type ijtag_function] [-source source_port_object [-inverted]]
```

Description

Specifies top design ports that will be added in the ICL file generated during ICL extraction. The ports can be of any ICL port type listed in the [ICL Port Types](#) table below.

Table 3-13. ICL Port Types

ICL Port Type	IJTAG Function	Direction
DataInPort	data_in	Input
DataOutPort	data_out	Output
ScanInPort	scan_in	Input
ScanOutPort	scan_out	Output
ClockPort	clock	Input
ToClockPort	to_clock	Output
ResetPort, ActivePolarity 1	reset	Input
ToResetPort, ActivePolarity 1	to_reset	Output
ResetPort, ActivePolarity 0	reset_n	Input
ToResetPort, ActivePolarity 0	to_reset_n	Output
TCKPort	tck	Input
ToTCKPort	to_tck	Output
TMSPort	tms	Input
ToTMSPort	to_tms	Output
TRSTPort	trst	Input
ToTRSTPort	to_trst	Output
CaptureEnPort	capture_en	Input
ToCaptureEnPort	to_capture_en	Output

Table 3-13. ICL Port Types (cont.)

ICL Port Type	IJTAG Function	Direction
ShiftEnPort	shift_en	Input
ToShiftEnPort	to_shift_en	Output
UpdateEnPort	update_en	Input
ToUpdateEnPort	to_update_en	Output
SelectPort	select	Input
ToSelectPort	to_select	Output
AddressPort	address	Input
WriteEnPort	write_en	Input
ReadEnPort	read_en	Input
ToIRSelectPort	to_ir_select	Output

You must specify the current design with the `set_current_design` command before issuing this command. Changing the current design will discard all previously specified input and output ICL ports. The tool validates that the ports specified as input ports are input or bidirectional ports, and ports specified as output ports are bidirectional or output ports.

Note

 You must load the design (gate-level Verilog or RTL) before you issue the `set_current_design` command as the tool verifies that the specified ports exist in the portlist. You may also load the top level design as a black-box module.

During ICL extraction, top level ports in the controlling fanin of input ports of sub-modules will automatically be inferred as input ports, and top level ports in the controlling fanout of output ports of sub-modules will automatically be inferred as output ports. You use this command if you want the top level ICL file to declare input or output ports even when they do not have any input or output ports in their fanin or fanout. Later on, when the current design is integrated into a higher module, ICL extraction will trace the newly created input and output ports and ijtag retargeting can be used to retarget iWrite and iRead commands to the boundary of the new top module.

The IJTAG function “`to_clock`” plays a special role. Ports of the current design with this IJTAG function serve as starting points for ICL Extraction tracing, and the tool enforces proper connectivity of those ports. If there is no appropriate driver for the newly created `ToClockPort` of the top module (`ToClockPort` of another ICL instance, `ClockPort` of the top module, internal clock), an I2 violation will be reported.

Arguments

- ***port_objects***

A required string that specifies a Tcl list of one or more design port names or a collection of one or more design ports. An error is generated if the specified ports are not found in the portlist of the design.

The ports in the collection must not have been used as arguments of the **-source** switch of another **add_icl_port** command before. (A port which is already in use as a “source port”, cannot be modified by means of **add_icl_ports**.)

- ***-type ijttag_function***

An optional switch and literal pair that specifies the function which the ports will have in the generated ICL module. The currently supported IJTAG functions are listed in the [ICL Port Types](#) table. If the **-type** option is not specified, the ICL port type will be inferred as DataInPort for input ports and DataOutPort for output ports. In case of bidirectional ports, the ICL port type is inferred as DataInPort, if the **-source** option is omitted, and as DataOutPort, if the **-source** option is specified. An error is generated if the specified type conflicts with the actual port direction.

- ***-source source_port_object***

An optional switch and value pair that specifies the top level port that will be used as source of the **port_objects** in the ICL file generated during ICL Extraction. This switch can be used to enforce feedthroughs (direct connections between a primary input port and a primary output port), which otherwise would not be created by ICL Extraction, because ICL Extraction does not start tracing at the top level ports.

The port specified as **source_port_object** must have been added as an ICL port by means of the **add_icl_ports** command before.

If the **ijttag_function** is **to_tck** or **to_ir_select**, then the **-source** specification is ignored, because TCKPorts and ToIRSelectPorts cannot have a “Source” property according to the IEEE 1687-2014 standard.

- ***-inverted***

An optional switch that specifies that the connections between the **source_port_object** and the **port_objects** shall be inverting.

Examples

Example 1

The following example declares an ICL DataInPort port D1, and three ICL DataOutPort ports O1 O2 and O3. The generated ICL module will include “DataInPort D1;”, “DataOutPort O1;”, “DataOutPort O2;” and “DataOutPort O3;” even if no ICL ports traced to or from them in the design.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
```

Example 2

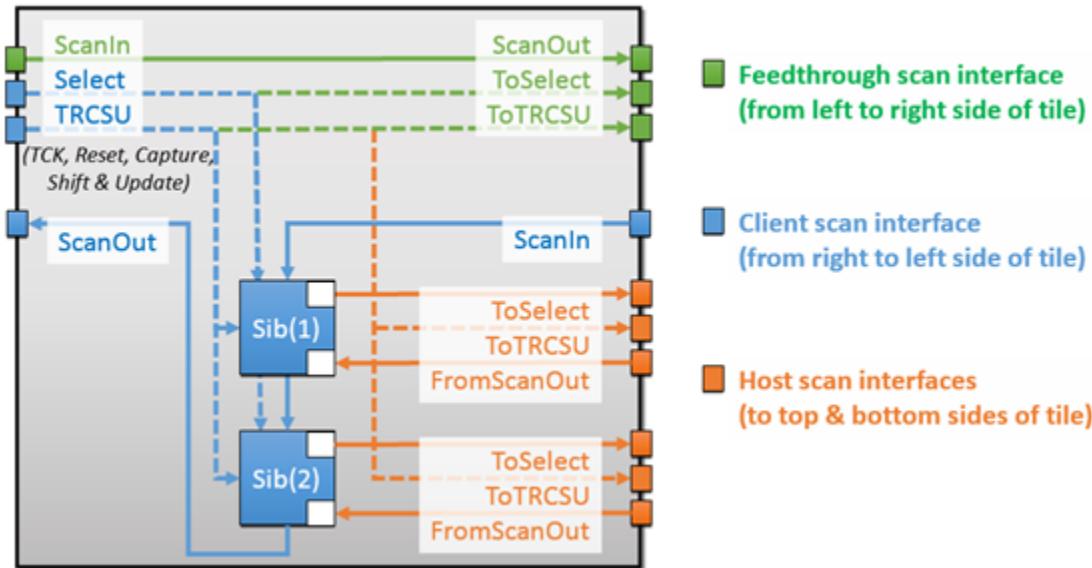
The following example declares two ICL host ScanInterfaces and an ICL client ScanInterface and enforces the control ports of the host ScanInterfaces to be driven directly by the control ports of the client ScanInterface. Parts of the SelectPorts, ToSelectPorts, ScanInPorts and ScanOutPorts are assumed to be created and connected automatically.

```
add_icl_scan_interfaces {host1 host2 client}
set_icl_scan_interface_ports -name host1 -ports {toCe1 toSe1 toUe1 toSel1 fromSo1 toSi1}
set_icl_scan_interface_ports -name host2 -ports {toCe2 toSe2 toUe2 toSel2 fromSo2 toSi2}
set_icl_scan_interface_ports -name client -ports {ce se ue sel si so}
add_icl_ports ce -type capture_en
add_icl_ports se -type shift_en
add_icl_ports ue -type update_en
add_icl_ports si -type scan_in
add_icl_ports {toCe1 toCe2} -type to_capture_en -source ce
add_icl_ports {toSe1 toSe2} -type to_shift_en -source se
add_icl_ports {toUe1 toUe2} -type to_update_en -source ue
add_icl_ports {toSi1 toSi2} -type scan_out -source si
```

Example 3

This example adds two SIBs inside a block and connects them into a ring between the si/so of the client interface. This same concept can be generalized to a greater number of scan interfaces if required. Two host ports are then created below each SIB to open up the path to for tiling rows.

Figure 3-19. Example Scan Interfaces



```
set_context dft -rtl
read_verilog ./RTL/coreb.v
set_current_design coreb

add_icl_ports ijtag_tck -type tck
set_design_level physical_block
set_system_mode analysis
```

add_icl_ports

```

// The below procedure creates & connects ports of scan interfaces host_top<n>
proc connect_top_SIBs {root_wrapper args} {
    for {set id 1} {$id <= 2} {incr id} {
        foreach port_name {ijtag_tck ijtag_reset ijtag_ce ijtag_se \
                           ijtag_ue ijtag_si} {
            set port [create_port to_${port_name}_top${id} -direction output]
            create_connections coreb_rtl_tessent_sib_${id}_inst/${port_name} $port
        }
        delete_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_from_so
        set port [create_port from_ijtag_so_top${id} -direction input]
        create_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_from_so $port
        set port [create_port to_ijtag_sel_top${id} -direction output]
        create_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_to_sel $port
    }
}
// The above callback procedure then gets invoked automatically after
// process_dft_spec
register_callback process_dft_specification.post_insertion connect_top_SIBs

// The below procedure declares a logical connection between feed_si and feed_so;
// It is very important, as the tools must trace through this logical connection
// later on, e.g. when generating patterns for a higher-level module
proc add_feedthrough_path { args } {
    # The next line can only work in 2017.4 or more recent versions
    # as it requires the ijtag_logical_connection attribute to be R/W in all modes
    set_attribute_value [get_current_design -icl] -name ijtag_logical_connection \
                        -value { {feed_si feed_so} }
    write_icl ./tsdb_outdir/dft_inserted_designs/coreb_rtl.dft_inserted_design/\
               coreb.icl -replace
}
// The above callback procedure then gets invoked automatically just before saving
// ICL
register_callback extract_icl.pre_write add_feedthrough_path

set dft [create_dft_spec]
// Let's just insert two SIBs here - the process can be generalized anyway
read_config_data -from_string {
    DftSpecification(coreb,rtl) {
        IjtagNetwork {
            HostScanInterface (client) {
                Sib(2) {
                }
                Sib(1) {
                }
            }
        }
    }
}
report_config_data $dft
process_dft_specification

// The add_icl_ports command is then used to declare the newly-added scan
// interface ICL ports
for {set id 1} {$id <= 2} {incr id} {
    add_icl_scan_interfaces host_top${id}
    foreach {port function} {reset to_reset_n sel to_select ce to_capture_en \
                           se to_shift_en ue to_update_en} {
        add_icl_ports to_ijtag_${port}_top${id} -type $function
}

```

```
}

add_icl_ports to_ijtag_si_top${id} -type scan_out
add_icl_ports from_ijtag_so_top${id} -type scan_in
set_icl_scan_interface_ports -name host_top${id} -ports [get_ports *_top${id}] \
    -allowed_no_source
}
extract_icl

set pat [create_patterns_spec]
report_config_data $pat
process_patterns_spec

run_testbench_simulations
check_testbench_simulations -report_status

exit
```

The following is the extracted ICL output for the coreb module in this example:

```

//-----
// File created by: Tessent Shell
// Version: 2017.4
// Created on: Mon Nov 27 13:43:32 PST 2017
//-----

Module coreb {
    // Created by ICL extraction
    ToTCKPort feed_tck;
    ScanInPort from_ijtag_so_top1 {
        Attribute connection_rule_option = "allowed_no_source";
    }
    ScanInPort from_ijtag_so_top2 {
        Attribute connection_rule_option = "allowed_no_source";
    }
    CaptureEnPort ijtag_ce;
    ResetPort ijtag_reset {
        ActivePolarity 0;
    }
    ShiftEnPort ijtag_se;
    SelectPort ijtag_sel;
    ScanInPort ijtag_si;
    ScanOutPort ijtag_so {
        Source coreb_rtl_tessent_sib_2_inst.ijtag_so;
    }
    TCKPort ijtag_tck;
    UpdateEnPort ijtag_ue;
    ToCaptureEnPort to_ijtag_ce_top1 {
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToCaptureEnPort to_ijtag_ce_top2 {
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToResetPort to_ijtag_reset_top1 {
        ActivePolarity 0;
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToResetPort to_ijtag_reset_top2 {
        ActivePolarity 0;
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToShiftEnPort to_ijtag_se_top1 {
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToShiftEnPort to_ijtag_se_top2 {
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToSelectPort to_ijtag_sel_top1 {
        Source coreb_rtl_tessent_sib_1_inst.ijtag_to_sel;
        Attribute connection_rule_option = "allowed_no_destination";
    }
    ToSelectPort to_ijtag_sel_top2 {
        Source coreb_rtl_tessent_sib_2_inst.ijtag_to_sel;
        Attribute connection_rule_option = "allowed_no_destination";
    }
}

```

```
}

ScanOutPort to_ijtag_si_top1 {
    Attribute connection_rule_option = "allowed_no_destination";
}
ScanOutPort to_ijtag_si_top2 {
    Source coreb_rtl_tessent_sib_1_inst.ijtag_so;
    Attribute connection_rule_option = "allowed_no_destination";
}
ToTCKPort to_ijtag_tck_top1 {
    Attribute connection_rule_option = "allowed_no_destination";
}
ToTCKPort to_ijtag_tck_top2 {
    Attribute connection_rule_option = "allowed_no_destination";
}
ToUpdateEnPort to_ijtag_ue_top1 {
    Attribute connection_rule_option = "allowed_no_destination";
}
ToUpdateEnPort to_ijtag_ue_top2 {
    Attribute connection_rule_option = "allowed_no_destination";
}
ScanInterface client {
    Port ijtag_ce;
    Port ijtag_reset;
    Port ijtag_se;
    Port ijtag_sel;
    Port ijtag_si;
    Port ijtag_so;
    Port ijtag_tck;
    Port ijtag_ue;
}
ScanInterface host_top1 {
    Port from_ijtag_so_top1;
    Port to_ijtag_ce_top1;
    Port to_ijtag_reset_top1;
    Port to_ijtag_se_top1;
    Port to_ijtag_sel_top1;
    Port to_ijtag_si_top1;
    Port to_ijtag_tck_top1;
    Port to_ijtag_ue_top1;
}
ScanInterface host_top2 {
    Port from_ijtag_so_top2;
    Port to_ijtag_ce_top2;
    Port to_ijtag_reset_top2;
    Port to_ijtag_se_top2;
    Port to_ijtag_sel_top2;
    Port to_ijtag_si_top2;
    Port to_ijtag_tck_top2;
    Port to_ijtag_ue_top2;
}
Attribute icl_extraction_date = "Mon Nov 27 13:43:32 PST 2017";
Attribute ijtag_logical_connection = " {feed_si feed_so} ";
Attribute created_by_tessent_icl_extract = "true";
Attribute tessent_design_id = "rtl";
Attribute tessent_design_level = "physical_block";
Attribute tessent_is_physical_module = "true";
Instance coreb_rtl_tessent_sib_1_inst Of coreb_rtl_tessent_sib_1 {
    InputPort ijtag_reset = ijtag_reset;
```

```

        InputPort ijttag_sel = ijttag_sel;
        InputPort ijttag_si = ijttag_si;
        InputPort ijttag_ce = ijttag_ce;
        InputPort ijttag_se = ijttag_se;
        InputPort ijttag_ue = ijttag_ue;
        InputPort ijttag_tck = ijttag_tck;
        InputPort ijttag_from_so = from_ijtag_so_top1;
        Attribute tesson_design_instance = "coreb_rtl_tesson_sib_1_inst";
    }
Instance coreb_rtl_tesson_sib_2_inst Of coreb_rtl_tesson_sib_1 {
    InputPort ijttag_reset = ijttag_reset;
    InputPort ijttag_sel = ijttag_sel;
    InputPort ijttag_si = coreb_rtl_tesson_sib_1_inst.ijtag_so;
    InputPort ijttag_ce = ijttag_ce;
    InputPort ijttag_se = ijttag_se;
    InputPort ijttag_ue = ijttag_ue;
    InputPort ijttag_tck = ijttag_tck;
    InputPort ijttag_from_so = from_ijtag_so_top2;
    Attribute tesson_design_instance = "coreb_rtl_tesson_sib_2_inst";
}
}

```

Related Topics

[create_dft_specification](#)

[delete_icl_ports](#)

[iRead](#)

[iWrite](#)

[set_current_design](#)

add_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Adds the specified scan interfaces to the new ICL top module that will be created during ICL extraction.

Usage

`add_icl_scan_interfaces name_list`

Description

Adds the specified scan interfaces to the new ICL top module that will be created during ICL extraction.

You only need to use this command if the top module on which ICL extract is running has more than one ScanInterface. In most cases, even when the top module has more than one ScanInterface, ICL extract is able to infer them correctly from the connections to and from the lower-level ICL modules. Only in complicated cases, the heuristic used by ICL extraction may fail; in this case, you use this command to describe the ScanInterface of the top module and they will be reflected in the generated ICL of the top module.

Every ScanInterface must have a port list, so after using the `add_icl_scan_interfaces` command, use the `set_icl_scan_interface_ports` command to set the list of ports for each ScanInterface.

You must set the current design with `set_current_design` before using the `add_icl_scan_interfaces` command.

Arguments

- ***name_list***

A required list that specifies the names of scan interfaces that ICL extraction will create in the new ICL top module.

Examples

In the following example, the `add_icl_scan_interfaces` command will create ScanInterfaces named TAP and Internal in the new ICL top module during ICL extraction. The `set_icl_scan_interface_ports` command will add the names of the ports to the TAP and Internal ScanInterfaces.

```
set_current_design tap1
add_icl_scan_interfaces {TAP Internal}
set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}
set_icl_scan_interface_ports -name Internal -ports {tdo tdrEn1 fromTdr1 ce se ue}
```

The resulting ICL model would look like this:

```
Module tap1 {  
  
    TCKPort      tck;  
    ScanInPort   tdi;  
    ScanOutPort  tdo      {  Source IRMux;  }  
    TMSPort      tms;  
    TRSTPort     trst;  
    ToSelectPort tdrEn1   {  Source sel1;  }  
    ScanInPort   fromTdr1;  
    ToCaptureEnPort ce;  
    ToShiftEnPort se;  
    ToUpdateEnPort ue;  
  
    ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}  
    ScanInterface Internal {Port tdo; Port tdrEn1; Port fromTdr1; Port ce;  
                           Port se; Port ue;}  
}
```

Note

 Every ScanInterface must have a port list. After using the `add_icl_scan_interfaces` command, use the `set_icl_scan_interface_ports` command to set the list of ports for each ScanInterface.

Related Topics

[delete_icl_scan_interfaces](#)
[get_icl_scan_interface_list](#)
[get_icl_scan_interface_port_list](#)
[set_current_design](#)
[set_icl_scan_interface_ports](#)

add_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables you to specify individual sites where the tool does not apply the Iddq restrictions.

Usage

`add_iddq_exceptions ZVAL_gate...`

Description

Enables you to specify individual sites where the tool does not apply the Iddq restrictions.

You can achieve maximum quiescence by using the appropriate IDDQ checks, and adding exceptions for as few sites as possible while still obtaining patterns that are not rejected for violation of IDDQ checks during fault simulation.

For more information on IDDQ, refer to “[IDDQ Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **ZVAL_gate...**

A required repeatable string that declares the gate name or gate identification number of a site for which the tool could report Iddq check violations. (See the [set_iddq_checks](#) command for specifics about setting Iddq checks). These sites are always roots of tri-state buses (ZVAL gates at the primitive level).

Note

 If a site does not have a unique name, you must provide a gate identification number.

The “report_gates -type zval” command gives a list of all candidates, but typically you only need to use the gate identification numbers reported by the [analyze_restrictions](#) command.

Examples

The following example adds Iddq restrictions, uses the [analyze_restrictions](#) command to locate the sources of violations, then adds an Iddq exception to remove one of them.

```
set_iddq_checks -int_float
simulate_patterns
analyze_restrictions
```

```
...
// Will attempt to locate and report specific -int_float violators.
// No single bus is constrained to Z (or already Z). Will analyze
// further.
// The following ZVAL gates cause -int_float to fail by themselves.
//   ZVAL gate /cpu/alu/U2392/ (54417). ABORTed.
//   ZVAL gate /cpu/alu/U4177/ (63712). ABORTed.
...
```

add_iddq_exceptions /cpu/alu/U2392/

To use the analyze_restrictions command most effectively, you must have design knowledge; in some cases, design knowledge is required in order to pinpoint the source of the violation.

Related Topics

[analyze_restrictions](#)

[report_gates](#)

[report_primary_outputs](#)

[set_iddq_checks](#)

add_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Creates a new logical connection path through the current design by specifying the source and destination of the logical path.

Usage

```
add_ijtag_logical_connection -from source_pin_or_port_name  
                          -to destination_pin_or_port_name [-inverted]
```

Description

Creates a new logical connection path through the current design by specifying the source and destination of the logical path.

During ICL connection tracing, logical connections will be followed in preference to the actual connections. When tracing reaches an instance pin with a logical connection, it will follow the logical connection across the design hierarchy. A forward trace will only follow logical connections that start on the pin being traced and then skip to the destination of the logical connection. A backward trace will only follow logical connections that have a destination on the pin being traced and then skip to the source of the logical connection.

See “[Example 1](#)” on page 284 for an example of a logical connection.

Use this command with care as it basically bypasses the structural tracing of the circuit and uses an act of faith on the behavior of the specified logical source. If the behavior turns out to be inconsistent with the specified logical sources, you will only detect the issue in simulation. Use this command to model logical sources through blackbox modules or to model a path where the logic is too complex. A circuit that gates TCK with capture and shift enable provides proper capture and shift behavior but would not be properly traceable by ICL extraction. In such a case, using a logical source to allow tracing through the tck gating logic is needed.

The following attributes are used to store the source...destination relationship for the two ends of the logical connection:

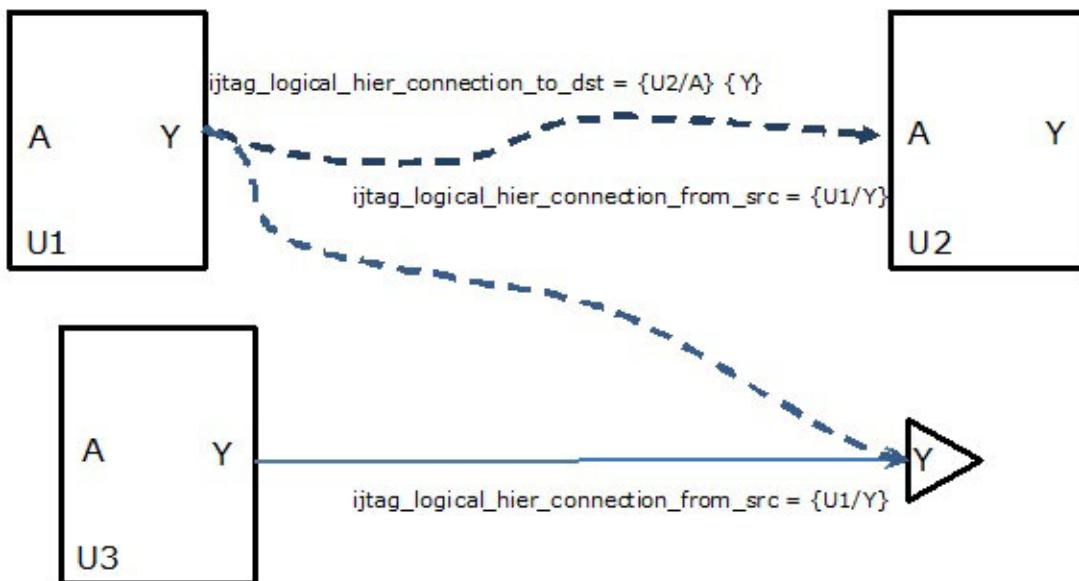
Table 3-14. Logical Connection Attributes

Attribute Name	Usage
ijtag_logical_hier_connection_from_src	<p>Associated with the top-level module port or instance pin that is the destination of a hierarchical logical connection.</p> <p>A string list of top level module port names and hierarchical instance pin path names to the source ends of these logical connections.</p> <p>This attribute is not inherited but does cause the hierarchical boundary to be preserved.</p>
ijtag_logical_hier_connection_from_src_inv	<p>Associated with the top-level module port or instance pin that is the destination of a hierarchical logical connection.</p> <p>A string list of top level module port names and hierarchical instance pin path names to the source ends of these logical connections.</p> <p>This attribute is not inherited but does cause the hierarchical boundary to be preserved.</p>
ijtag_logical_hier_connection_to_dst	<p>Associated with the top-level module port or instance pin that is the source of a hierarchical logical connection.</p> <p>A string list of top level module port names and hierarchical instance pin path names to the destination ends of these logical connections.</p> <p>This attribute is not inherited but does cause the hierarchical boundary to be preserved.</p>

Table 3-14. Logical Connection Attributes (cont.)

Attribute Name	Usage
ijtag_logical_hier_connection_to_dst_inv	<p>Associated with the top-level module port or instance pin that is the source of a hierarchical logical connection.</p> <p>A string list of top level module port names and hierarchical instance pin path names to the destination ends of these logical connections.</p> <p>This attribute is not inherited but does cause the hierarchical boundary to be preserved.</p>

The following figure illustrates the usage of these attributes.

Figure 3-20. Logical Connection Attributes

These attributes are available for direct read only introspection by the user, like the other predefined ICL extraction attributes. They will all be cleared when the `set_current_design` command is executed.

Arguments

- **-from source_pin_or_port_name**

A required switch and value pair that specifies the source point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to destination_pin_or_port_name**

A required switch and value pair that specifies the destination point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-inverted**

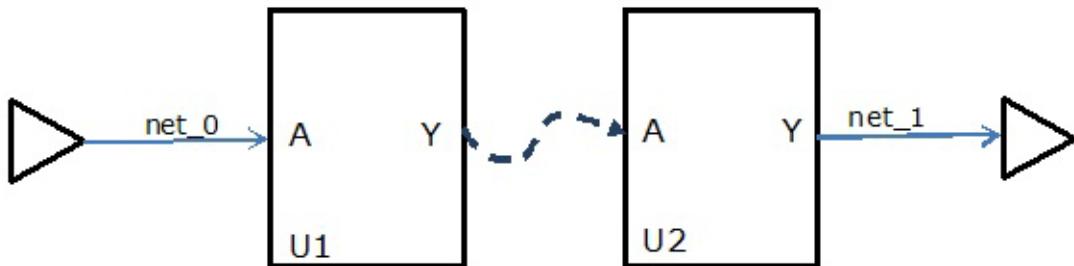
An optional switch that specifies that the logical connection is inverted.

Examples

Example 1

The following example illustrates a logical connection from pin U1/Y to pin U2/A. A forward trace that arrives at pin U1/Y will continue on from pin U2/A. A backward trace that arrives at pin U2/A will continue on from pin U1/Y.

Figure 3-21. Logical Connection Example



At the netlist level all interconnections start at an instance pin or a top level module port and end at an instance pin or top level module port. Establishing a logical connection within the design hierarchy will not alter the way that design hierarchy is traversed by ICL extraction tracing when that design hierarchy is used as part of a larger design hierarchy, from a different top-level module.

Example 2

The `add_ijtag_logical_connections` command will create logical connections between the pins /i2/in and /i2/out, /i3/in and /i3/out and i4/in and /i4/out, and an inverted logical connection between /i5/in and /i5/out. The `report_ijtag_logical_connections` command will report all logical connections in the current design.

add_ijtag_logical_connection -from /i2/in -to /i2/out

```
add_ijtag_logical_connection -from /i3/in -to /i3/out
add_ijtag_logical_connection -from /i4/in -to /i4/out
add_ijtag_logical_connection -from /i5/in -to /i5/out -inverted
report_ijtag_logical_connections

// IJTAG Logical Connections:
// From Source To Destination Inversion
// ====== ====== ======
// /i2/in    /i2/out      no
// /i3/in    /i3/out      no
// /i4/in    /i4/out      no
// /i5/in    /i5/out      yes
```

Related Topics

[delete_ijtag_logical_connection](#)
[report_ijtag_logical_connections](#)

add_input_constraints

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Constrains primary inputs to specified values.

Usage

Context: dft -scan and dft -test_points

```
add_input_constraints
{{[-AL] | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
 constraint_switch} |
{[-EQuivalent reference_pin [equivalent_pin]... [-INvert inverted_pin...]]
 [constraint_switch]}}
```

Context: patterns -scan

```
add_input_constraints
{{[-AL] | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
 constraint_switch} |
{[-EQuivalent reference_pin [equivalent_pin]... [-INvert inverted_pin...]]
 [constraint_switch]}}
[-HOld] [-NO_Z] [-SLow_pad [-CELL model_name]]
```

Context: patterns -ijtag

```
add_input_constraints
{{[-AL] | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude]
 constraint_switch} |
{[-EQuivalent reference_pin [equivalent_pin]... [-INvert inverted_pin...]]
 [constraint_switch]}}
```

Description

Constrains primary inputs to specified values.

The add_input_constraints command constrains primary input pins to specified values.

Note

 Do not force a constrained pin to a new value in any cycle of a named capture procedure (NCP), even temporarily. To ensure state stability, a constrained pin's state must be kept constant for all cycles of NCPs.

In the patterns -ijtag context, you can constrain all Verilog ports unless the port is also an ICL port. Input constraints on ICL ports are allowed only for ICL ports of type TRSTPort and ClockPort. For more information, see “[How to Constrain Inputs](#)” in the *Tessent IJTAG User’s Manual*.

Inferred input constraints

Some input constraints are automatically applied to certain input ports of the current design, depending on the current context and the ICL port function. There are also attributes that trigger the creation of input constraints.

Constraints derived from ICL port functions

If there is an ICL description for the current design, then the following constraints are applied during the processing of the command `set_current_design` in all contexts except “patterns -ijtag”: A port with ICL port function TMSPort is constrained to C0. A port with ICL port function TRSTPort is constrained to C1 unless there is already a CT1 constraint for that port. Ports with ICL port function ResetPort are constrained to their inactive values, using the constraint types “C0” or “C1”. All ports with ICL port functions CaptureEnPort, ShiftEnPort and UpdateEnPort are constrained to C0, unless the associated ICL ScanInterfaces contain ports with ICL port function SelectPort, in which case only the SelectPorts are constrained to C0. When the context is set to “patterns -ijtag”, the constraints are automatically removed. They are reapplied if the context is set to any other context but “patterns -ijtag”.

Constraints derived from ICL module attributes

If the ICL module, which is associated with the current design, contains the “forced_low_input_port_list” and/or “forced_high_input_port_list” `icl_module` attributes, then the ports referenced by those attributes are constrained appropriately, using the constraint types “C0” and “C1”. The constraints are applied during the processing of the command “`set_current_design`”.

Constraints derived from port attributes

Ports with the `Port` attribute “function” set to “power” or “ground” are constrained appropriately, using the constraint types “CT1” and “CT0”. The constraints are applied during the processing of the command “`set_current_design`”.

Constraints on ports with ICL port function DataInPort

At the end of a `test_setup` procedure that contains `iCall` or `iMerge` statements, all ports with ICL port function DataInPort are constrained to C0 or C1, according to the last values that have been assigned to them during the IJTAG retargeting of the `iCall` and `iMerge` commands. This behavior is suppressed, if the port has the `icl_module` attribute “`tessent_no_input_constraints`” set to “on”.

Arguments

- -ALL

An optional switch that applies the constraint to all applicable primary input pins except clocks, read controls, and write controls. If `-Slow_pad` is also specified, the constraint is only applied to primary input pins that can take a Z value. This is the default.

- *primary_input_pin_name*

An optional, repeatable string that specifies the name of a primary input pin to which to apply the constraint.

- **-BIDI_Only**
An optional switch that specifies to apply the constraint to bidirectional pins only.
- **-BIDI_Exclude**
An optional switch that specifies to apply the constraint only to primary input pins that are not bidirectional.
- **-SCAN_INputs**
An optional switch that specifies to apply the constraint to scan input pins only.
- **-EQuivalent *reference_pin* [*equivalent_pin*]...**
An optional switch, string, and repeatable string triplet that specifies for the *primary_input_pin_name*(s) to take on the value of the *reference_pin*. If you optionally specify one or more of the *primary_input_pin_name*(s) as *equivalent_pin*(s), those PIs in the *equivalent_pin* list take on the value of the *reference_pin*. You must list pins you want to be equivalent in value to the reference pin before the -Invert argument when used.
- **-INvert *inverted_pin*...**
An optional switch and repeatable string that together with the -Equivalent switch specify for the primary input pins, *inverted_pin*, to take on the inverted value of the *reference_pin*. You must list the pins you want to be inverted in value after the -Invert argument.
- ***constraint_switch***
A required switch that specifies the constant value to which to constrain the primary input pin(s). This switch is required unless you specify the -EQuivalent switch.

The choices, from which you can specify only one, are as follows:

-C0 — Specifies application of the constant 0 to the chosen primary input pin.

The tools do not use constant 0 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use a -C0 constraint if the constraint is valid in test mode, but *not* valid in system mode.

Note

 If the constraint is valid in system mode as well, consider using -CT0 instead. Faults associated with tied circuitry resulting from a -CT0 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C0 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT0 only for valid system mode constraints.

-C1 — Specifies application of the constant 1 to the chosen primary input pin.

The tools do not use constant 1 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use -C1 if the constraint is valid in test mode, but *not* valid in system mode.

Note

 If the constraint is valid in system mode as well, consider using -CT1 instead. Faults associated with tied circuitry resulting from a -CT1 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C1 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT1 only for valid system mode constraints.

-CX — Specifies application of the constant X (unknown) to the chosen primary input pin. The tools do not use the specified primary input pin for control.

-CZ — Specifies application of the constant Z (high-impedance) to the chosen primary input pin. The CZ constraint is only applicable to bidi pin and PI that fans out only to Z consumer gates, such as PO, or bus. For a PI that fans out to non Z consumer gate, when CZ is added the tool will issue a warning message during the flattening process and convert CZ constraint into CX constraint automatically. For example:

```
// Warning: Invalid constant Z pin constraint on pin te5 changed to
// constant X.
```

-CT0 — Specifies application of the constant TIE0. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT0 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are *not* included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C0 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT0 instead of a -C0 constraint; but take care that any -CT0 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

 You cannot override a -CT0 value (from a test procedure file for example). If you need this flexibility, use -C0 instead.

-CT1 — Specifies application of the constant TIE1. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT1 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C1 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT1 instead of a -C1 constraint; but take care that any -CT1 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

 You cannot override a -CT1 value (from a test procedure file for example). If you need this flexibility, use -C1 instead.

-CTZ — Specifies application of the constant TIEZ. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CTZ pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -CZ pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CTZ instead of a -CZ constraint; but take care that any -CTZ constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

- **-HOld**

An optional switch that, for clock sequential patterns only, holds the constant constraint value at the primary input(s) for all cycles of a *clock_sequential* or *multiple_load* pattern. Even if a pattern has multiple loads, the primary input's value cannot change among different load intervals. The hold does not apply to clocks, read controls and write controls. It also does not apply to launch off shift patterns, RAM sequential patterns, or patterns with a clock procedure.

If the fault type is set to transition and you enter a “*set_transition_holdpi On*” command, the constraint defined by the *add_input_constraints* command is overridden by the *set_transition_holdpi* command; that is, all primary inputs except clocks and read/write controls are going to hold their values even if you have not applied a constraint to a particular primary input. Similar behavior occurs when the fault type is set to path delay and you enter a “*set_pathdelay_holdpi On*” command.

- **-NO_Z**

An optional switch that specifies not to allow a Z value at the specified primary input pin(s) during test generation, even if the pins can support a Z value. During good machine simulation, a pattern is rejected if it assigns a Z value to any primary input that is specified to have no Z value. This switch is invalid with either the -CZ or -CTZ switch.

- **-Slow_pad**

An optional switch that specifies to X out the value at a bidirectional pin if the pin is driven from within during test generation.

-CELL *model_name*

An optional switch and string pair that together with the -Slow_pad switch specifies the scope of the bidirectional pins to be included as slow pads. Only the bidirectional pins that drive the instances initiated from the specified library cell name are added as slow pads.

Examples

Example 1

The following example shows two differential clocks, CLK1 off state 0 and CLK2 off state 1. Since the pins are equivalent, you only need to add the clock to one of the pins. The command

order is important: **you must** first define equivalent pins, then define one of them as the clock, otherwise the tool will issue an error. The tool will infer CLK2 to be a clock.

```
add_input_constraints -equivalent CLK1 -invert CLK2
add_clocks 0 CLK1
// Note: Equivalent pin CLK2 also added to clock list.
```

Example 2

The following example shows how to add equivalent clocks that are inverted.

```
SETUP> add_clocks 0 mode[1] -pulse_always
SETUP> add_clocks 1 mode[2] -pulse_always
SETUP> report_clocks
User-defined Clocks (2):
=====
Sync and Async Source Clocks
=====
-----  -----  -----  -----
Name      Off State   Constraints Internal
-----  -----  -----  -----
mode [2]    1        Pulse always    No
mode [1]    0        Pulse always    No
SETUP> add_input_constraints -equivalent mode[1] -inv mode[2]
SETUP> report_input_constraints
// Primary Input  Equivalent  C-Type  Hold      No-Z      Slow-Pad
// -----  -----  -----  -----  -----  -----
// mode [1]          NONE      No     Undetermined      No
//                   ~mode [2]    NONE      No     Undetermined      No
```

Example 3

This example specifies multiple equivalent pins: clk, clk1, and clk2. Pin clk1 is equivalent to the reference pin, clk. Pin clk2 is inverted from the reference pin, clk.

```
SETUP> add_input_constraints -equivalent clk clk1 -invert clk2
SETUP> add_clocks 0 clk
// Note: Equivalent pin clk1 and clk2 are also added to clock list.
SETUP> report_clocks
```

User-defined Clocks (3) :

Sync and Async Source Clocks					
Name	Label	Off State	Constraints	Internal	
'clk'	clk	0		No	
'clk2'	clk2	1		No	
'clk1'	clk1	0		No	

Related Topics

[delete_input_constraints](#)

[report_input_constraints](#)

add_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB.

Usage

```
add_layout_core_information {-def list_of_DEF_files | -deflist tcl_list |
    {-design top_design_name -die_area {die_area} -instance instance -placement {x y}
     -orientation orientation} }
```

Description

In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB. This information can come from three sources, which are mutually exclusive: a space-separated list of top-level DEF files, a Tcl list of top-level DEF files, or the actual core instance data chip-mapped to the top level.

To use this command you must have a valid LDB that you previously opened using the open_layout command. Normally, you also have top-level DEF files that instantiate the core represented by the currently opened LDB, and the DEF file for the core represented by the currently opened LDB. If you only have core-level DEF files, you can manually add chip-mapped core instance information.

When executed, the command extracts the instance information for the core represented by the LDB from the top-level DEF file(s) and stores this information in the LDB. This makes the LDB instance-aware, which is a requirement for hierarchical layout-aware diagnosis.

This command allows you to store core instance information from multiple SoC designs in the same LDB, which is helpful when the same core is used in multiple designs.

See “[Diagnosis for Hierarchical Design](#)” in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- **-def list_of_DEF_files**

A switch and string pair that specifies a space-separated list of top-level DEF files that contain placement and orientation information for each core.

- **-deflist tcl_list**

A switch and string pair that specifies a TCL list of top-level DEF files that contain placement and orientation information for each core.

- **-design *design_name* -die_area {*die_area*} -instance *instance* -placement {*x y*}**
-orientation *orientation*

A series of switches that must be specified together. Use these switches to manually add the instance information for a core to an existing LDB. You must specify separate add_layout_core_information commands for each core you want to manually add.

- design *top_design_name*: The name of the top module of the design.
- die_area {*die_area*}: Four integers that specify the X,Y coordinates of the core with origin at 0,0. The first and second integers specify the bottom-left corner of the core, and the third and fourth integers specify the top-right corner of the core.
- instance *instance*: The core instance name.
- placement {*x y*}: Integers that represent the x, y placement of the specified core in the top-level DEF, where x and y indicate the bottom right corner of the core.
- orientation *orientation*: A string that specifies the orientation of the core instance, where the possible choices are: { N | W | S | E | FN | FW | FS | FE }.

Examples

The following example adds two core instances of coreA to an existing LDB.

```
add_layout_core_information top_chip -die_area {-4000 -4000 20000000 40000000} \
    -instance cpu_coreA -placement {2000 2000} -orientation N
add_layout_core_information top_chip -die_area {-2000 -2000 10000000 20000000} \
    -instance cpu_coreA -placement {50 100} -orientation FS
```

Related Topics

[delete_layout_core_information](#)
[report_layout_core_information](#)

add_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).

Usage

`add_lfsr_connections pin.pathname lfsr_name position...`

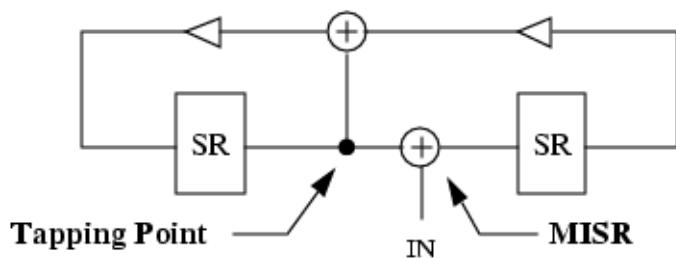
Description

Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).

The `add_lfsr_connections` command connects a core logic pin to an LFSR. You specify this pin with the *pin.pathname* argument. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes that the output of the 0 bit position connects to the input of the highest bit position. If you select multiple bits of a Pseudo-Random Pattern Generator (PRPG) for the *position* argument, the tool assumes they are all exclusive-ORed together to create the value for the pin.

If you determine that multiple pin names must connect to a bit position of a Multiple Input Signature Register (MISR), you must issue a separate `add_lfsr_connections` command for each pin. The tool assumes the pins are all exclusive-ORed together to create the value for the next MISR input. The tool also assumes that the physical placement of the MISR connections is after the tapping points as shown in [Figure 3-22](#).

Figure 3-22. MISR placement



You can use the `report_lfsrs` command to display all the LFSRs with their current values and tap positions.

Arguments

- *pin.pathname*

A required string that specifies the pin pathname (*<instance pathname>/<pinname>*) of the core logic pin that you want to connect to the LFSR specified by *lfsr_name*.

- ***lfsr_name***

A required string that specifies the name of the LFSR to which you want to connect the *pin_pathname*.

- ***position***

A required repeatable integer that specifies the bit positions of the *lfsr_name* at whose outputs you want to place connections. A bit position is an integer number, where 0 indicates the least significant bit position. The tool assumes the output of the 0 bit position connects to the input of the highest bit position.

Examples

The following example connects an LFSR to a scan-in pin and another LFSR to a scan-out pin:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
add_lfsr_connections scan_in.1 lfsr1 2
add_lfsr_connections scan_out.0 misr1 3
```

Related Topics

[add_lfsrs](#)

[add_lfsr_taps](#)

[delete_lfsr_connections](#)

[report_lfsr_connections](#)

add_lfsr_taps

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).

Usage

`add_lfsr_taps lfsr_name position...`

Description

Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).

The `add_lfsr_taps` command sets the tap configuration of an LFSR. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes the output of the 0-bit position connects to the selected tap points and that the 0-bit position cannot itself be a tap point.

If the LFSR is defined as a CA, the tap positions define the position of the self-feedback cells (those CA cells defined using rule 150). At least one tap must be defined; otherwise, the consistency check of the PRPG issues an error message.

You can use the `report_lfsrs` command to display all of the LFSRs with their current values and tap positions. You can change the default setting of the `tap_type` switches by using the `set_lfsrs` command.

Arguments

- ***lfsr_name***

A required string that specifies the name of the LFSR on which you want to place the taps.

- ***position***

A required, repeatable integer that specifies the bit positions of the `lfsr_name` at whose outputs you want to place the taps or, for CAs, the bit positions of the self-feedback cells.

Examples

Example 1

The following example places taps on the newly-added LFSRs:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```

Example 2

The following example defines a 32-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, 23, 25, and 30:

```
add_lfsrs my_prpg prpg 32 FFFFFFFF -CA  
add_lfsr_taps my_prpg 3 7 12 15 23 25 30
```

Related Topics

[add_lfsrs](#)

[delete_lfsr_taps](#)

[report_lfsrs](#)

[set_lfsrs](#)

[add_lfsrs](#)

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Adds LFSRs for use as PRPGs or MISRs.

Usage

```
add_lfsrs lfsr_name {Prpg | Misr} length seed [-Both | -Serial | -Parallel]  
[-Out | -In | -Ca [-INCrement_step value]]
```

Description

Adds LFSRs for use as PRPGs or MISRs.

The add_lfsrs command defines LFSRs, which the tool uses as PRPGs, to create pseudorandom values for the BIST patterns or as MISRs to compact responses.

You specify the LFSR's shift technique by using one of the following shift_type switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following tap_type switches: -Out or -In; or you specify the type of LFSR as cellular automata using the tap_type switch -Ca. You can change the default setting of the shift_type and tap_type switches by using the set_lfsrs command.

Arguments

- ***lfsr_name***

A required string that specifies the name that you want to assign to the LFSR. Note that you cannot use the keyword “all” for this name.

- **Prpg**

A literal that indicates the LFSR functions as a PRPG.

- **Misr**

A literal that indicates the LFSR functions as a MISR.

- ***length***

A required integer, greater than 1, that specifies the number of bits in the LFSR.

- ***seed***

A required, right-justified, hexadecimal number, greater than 0, that specifies the initial state of the LFSR.

The following lists the shift_type switches of which you can choose only one.

- **-Both**

An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default unless you change it with the set_lfsrs command.

- **-Serial**
An optional switch specifying that the LFSR shifts serially the number of times equal to the length of the longest scan chain for each scan pattern.
- **-Parallel**
An optional switch specifying that the LFSR parallel shifts once for each scan pattern.
The following lists the three tap_type switches of which you can only choose one.
- **-Out**
An optional switch specifying that the exclusive-OR taps reside outside the register path. This is the default unless you change it with the set_lfsrs command.
- **-In**
An optional switch specifying that the exclusive-OR taps reside in the register path.
- **-CA**
An optional switch specifying that the PRPG is of type *cellular automata*. This switch is only valid if the LFSR type is specified as *Prpg*.
- **-INCrement_step value**
An optional switch that specifies the number of times the PRPG is advanced for each shift cycle. The *value* argument is a required integer.

Examples

Example 1

The following example defines an LFSR to be a PRPG and another LFSR to be a MISR:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```

Example 2

The following example defines an LFSR to be a PRPG without a phase shifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Example 3

The following example defines an LFSR to be a PRPG with a phase shifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsr_connections scanin_1 my_prpg 0 3 5
add_lfsr_connections scanin_2 my_prpg 1 7 11
...
add_lfsr_connections scanin_24 my_prpg 9 15 23
```

Example 4

The following example defines an LFSR to be a PRPG and specifies to advance the PRPG eight times for each shift cycle.

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA -inc 8
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Related Topics

[add_lfsr_taps](#)
[add_lfsr_connections](#)
[delete_lfsrs](#)
[report_lfsrs](#)
[set_lfsrs](#)
[set_lfsr_seed](#)

add_lists

Context: dft -edt, patterns -scan

Mode: analysis

Adds pins to the list of pins on which to report.

Usage

`add_lists pin.pathname...`

Description

Adds pins to the list of pins on which to report.

In “dft -edt” context, you cannot report values on gates in the EDT logic with this command because values inside the EDT logic are not always accurate.

Tip

This command is a useful debugging aid, as it provides a way to check the values simulated for specific output pins of cells within the design.

To list all the pins on which the tool will currently report during simulation, use the [report_lists](#) command. To turn off reporting of a pin’s simulation values, use the [delete_lists](#) command. To direct the tool to write the pins’ simulation values to a file instead of displaying them on screen, use the [set_list_file](#) command.

When switching to Setup mode, the tool discards all pins from the report list.

Arguments

- *pin.pathname*

A required, repeatable string that specifies the output pins on which to report during good and/or faulty machine simulation.

Examples

The following example writes to the file, *pinlist_goodsim.txt*, the values simulated for two output pins during simulation of an external pattern source:

```
set_system_mode analysis
read_patterns pattern1
add_lists /i_1006/o /i_102/o
set_list_file pinlist_goodsim.txt -replace
simulate_patterns
```

The following shows the format of the resulting list file: the first column lists the pattern index of each pattern simulated, the second column shows the value simulated on each monitored pin for each pattern. Note that the pattern index refers to the simulated parallel patterns rather than the effective patterns that will be stored after ATPG.

```
//      /i_1006/o
//      |/i_102/o
//
//      ||
//-----//
//      0  11
//      1  10
//      2  00
//      3  10
//      4  01
//      5  01
...

```

Notice that in analysis system mode, the report shows the good value of the pin. The tools also shows the values at each test pattern.

The following is example list file content for the same two pins when simulated in analysis system mode:

```
//      /i_1006/o
//      |/i_102/o
//
//      ||
//-----//
//      0  11
//      0  0- { /i_1006/o@0 }
//      1  10
//      1  0- { /i_1006/o@0 }
//      2  00
//      2  1- { /i_1006/o@1 }
//      3  10
//      3  0- { /i_1006/o@0 }
//      4  01
//      4  1- { /i_1006/o@1 }
//      5  01
//      5  1- { /i_1006/o@1 }
...

```

The report now includes two lines of information for each pattern simulated. The first line shows the good machine value of each pin as before. The second line shows the faulty value of each pin if different from the good value (a “-” indicates the faulty value was the same as the good value). Inside the “{}” is the fault site and its faulty machine stuck-at value the tool simulated to produce the faulty machine results.

Related Topics

[delete_lists](#)
[report_lists](#)
[set_list_file](#)

add_loadboard_loopback_pairs

Context: patterns -ijtag, -scan, -scan_diagnosis, -scan_retargeting

Modes: Setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Creates a loopback connection from specified output ports to specified input ports.

Usage

```
add_loadboard_loopback_pairs -inputs {input_object_spec} -outputs {output_object_spec}  
[-ac_delay_to_z ac_delay_to_z]
```

Description

This command creates a loopback connection from specified output ports to corresponding specified input ports. An ac_delay time can be specified for the connections. This command provides direct connections of the DC and the CouplingCapacitor connections in the simulation testbenches of custom pattern (using open_pattern_set, close_pattern_set, and write_patterns).

Arguments

- **-inputs** {*input_object_spec*}

A required switch and string pair that specifies the input ports for a loopback connection.

The *input_object_spec* string can specify one or more input ports that get paired with an output port specified in the -outputs *output_object_spec* switch. If the input port count does not match the output port count, the tool issues an error.

- **-outputs** {*output_object_spec*}

A required switch and string pair that specifies the output ports for a loopback connection. The *output_object_spec* string can specify one or more output ports. The output ports specified get paired with an input port specified in the -inputs *input_object_spec* switch. If the output port count does not match the input port count, the tool issues an error.

- **-ac_delay_to_z** *ac_delay_to_z*

An optional switch and value pair that specifies the ac-delay-to-z value for all the loopback connections made in by this command. The value must be a valid time specification such as 10ns. This value triggers the creation of a coupling capacitor that will tri-state (Z) the signal going back to the input after the amount of time on each level transition of the output signal.

Examples

This example shows how this command is used to create a loopback pairs between two outputs and inputs. An ac_delay of 10ns is specified. The first output, dout[5], listed in the -outputs switch is connected to the first input, din[4], listed in the -inputs switch:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]} \  
-ac_delay_to_z 10ns  
report_loadboard_loopback_pairs
```

```
-----  
Inputs   Outputs   AC Time to Z  
-----  
din[4]   dout[5]  10ns  
ue       dout[0]  10ns  
-----
```

write_patterns pat.v -pattern_sets pat1 -verilog

The following statements are added to the Verilog file.

```
wire to_din_4;  
assign to_din_4 = \din[4];  
CouplingCapacitor #(10.000000) din_4_CC_INST(.in(\dout[5]),  
.out(to_din_4));  
  
wire to_ue;  
assign to_ue = ue;  
CouplingCapacitor #(10.000000) ue_CC_INST(.in(\dout[0]), .out(to_ue));
```

[add_misrs](#)

Context: patterns -scan

Mode: setup, analysis

Specifies the MISR structure and initial seed.

Usage

```
add_misrs misr_name [-size integer] [-seed hexadecimal] [-taps index...]
```

Description

Specifies the MISR structure and initial seed.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

This command is included in the LogicBIST fault simulation dofile to describe the MISR logic synthesized.

Arguments

- ***misr_name***

A required string that specifies the name of the MISR. The ***misr_name*** is subsequently used by the tool for referring to this MISR for specifying connections from scan chains, and in reporting.

- **-size *integer***

A required switch and integer that specifies size of the MISR.

- **-seed *hexadecimal***

An optional switch and hexadecimal number pair that specifies the initial seed. Use this command if you intend on performing fault simulation with a different starting MISR seed.

- **-taps *index***

An optional switch and repeatable integers that specifies the tap points for the MISR feedback network. The tap index starts from 1 to be consistent with the decompressor used as a PRPG in the LogicBIST mode and associated decompressor commands.

Examples

This example shows the add_misrs command in context patterns -scan, mode setup. This example adds a 24-bit MISR named “misr” with initial seed of all 0-s:

```
add_misrs misr -size 24 -seed 000000 -taps 21 22 24
```

Related Topics

[delete_misrs](#)

[report_misr_connections](#)
[report_misrs](#)
[set_misr_connections](#)

add_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Places nofault settings either on pin pathnames, pin names of specified instances, or modules.

Usage

```
add_nofaults {{modulename ... -Module [-PIN module_pin_pathname ...] |  
          object_expression ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}] [-Keep_boundary]} | -Wire  
          [-VERBose] [(> | >>) file_pathname]
```

Description

Places nofault settings either on pin pathnames, pin names of specified instances, or modules.

By specifying pathnames of pins, instances, or modules while in Setup mode, the add_nofaults command places a nofault setting either on the specific pins or on boundary and internal pins of the instances or modules. All added nofault pin pathnames are in the user class.

Note

 All switches except -Wire are valid only before flattening. The -Wire switch is valid only after flattening.

The -Stuck_at switch applies to either stuck-at faults or transition faults. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a value, then the tool places a nofault setting on both stuck-at (transition) values. If you add faults with the add_faults command after you issue the add_nofaults command, the specified pin pathnames or boundary and internal pins of instances or modules cannot be sites for those added faults.

Note

 When you add nofault settings, the tool deletes the flattened simulation model if it exists. This removes information you added after model flattening, such as ATPG functions. To avoid losing this information or having to wait for the tool to flatten the design again, add nofault settings prior to model flattening.

Arguments

- **modulename**

A repeatable string that specifies the name of a module to which you want to assign nofault settings. You must include the -Module switch when you specify a module name.

- **-Module[-PIN *module_pin_pathname* ...]**

A switch that specifies to interpret the **modulename** argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?)

wildcards for the ***modulename*** argument, and the tool adds the nofault for all matching modules or library models.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given module name and adds nofaults for every pin specified in module_pin.pathname list.

- ***object_expression***

A string representing a list of pathnames of instances or pins for which you want to assign nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- **-PIN**

An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then assign nofault settings to all the pins matched.

- **-Instance**

An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then assign nofault settings to all boundary and internal pins of the instances matched (unless you use the -Keep_boundary switch).

- **-Stuck_at 0 | 0 | 1**

An optional switch and literal pair that specifies to which stuck-at or transition values you want to assign a nofault setting. The choices are as follows:

01 — A literal that specifies to place a nofault setting on both the “stuck-at-0” and “stuck-at-1” faults for stuck-at faults; or on both the “slow-to-rise” and “slow-to-fall” faults for transition faults. This is the default.

0 — A literal that specifies to place a nofault setting on only the “stuck-at-0” faults for stuck-at faults; or on only the “slow-to-rise” faults for transition faults.

1 — A literal that specifies to place a nofault setting on only the “stuck-at-1” faults for stuck-at faults; or on only the “slow-to-fall” faults for transition faults.

- **-Keep_boundary**

An optional switch that specifies to apply nofault settings to the inside of the specified instance/module, but allow faults at the boundary pins of these instances/modules. This option does not apply to nofaults on pin pathnames.

- **-Wire**

A switch that specifies for the tool to no-fault a cone of logic fanning out from a stem forward into a single wire gate. The tool no-faults the entire cone of logic between the stem and the wire gate, but not the stem or the output of the wire gate. There must be no other fanout from this cone of logic. This switch is valid only after flattening.

Note



This switch affects all qualifying logic cones.

- **-VERbose**

By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

```
// Note: Addingnofaultsfor330faultsites.
```

When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- **>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- **>>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example defines nofault settings for all the pins in a particular instance, so when you add all faults to the circuit for an ATPG run, the tool will not place faults on the pins of that instance:

```
add_nofaults i_1006 -instance
set_system_mode analysis
create_patterns
```

Related Topics

[add_faults](#)
[delete_faults](#)
[delete_nofaults](#)
[report_faults](#)

report_nofaults

add_nonscan_instances

Context: dft -no_rtl, dft -rtl, dft -scan, dft -test_points

Mode: setup, analysis

Sets a user-specified is_non_scannable attribute on the specified design objects.

Usage

Context: dft -scan, dft -test_points

```
add_nonscan_instances { -INStances instance_spec | -Modules module_spec |
    -scan_elements scan_element_spec | -Control_signals pin_spec | -All } [-force]
```

Context: dft -no_rtl, dft -rtl

```
add_nonscan_instances { -INStances instance_spec | -Modules module_spec |
    -scan_elements scan_element_spec | -Control_signals pin_spec |
    -rtl_reg net_spec | -All } [-force]
```

Description

Sets a user-specified is_non_scannable attribute on the specified design objects, thus excluding them from the scan insertion process.

The add_nonscan_instances command sets the user-specified is_non_scannable attribute on the specified instances, all sequential cells within the instances of the specified modules, the specified scan elements, the instances controlled by the specified control signals, or all instances.

Also, the tool will not perform the scannability rule checks on non-scan instances.

When adding non-scan instances with control signals, the tool adds all sequential instances controlled by the control signals to the non-scan instance list. Control pins can be any clock, set, or reset signal (primary input) that controls the contents of sequential instances. You can use the [report_control_signals](#) command to see the pins that control the sequential instances in the design.

If TIE0 and TIE1 non-scan cells are scannable, they are considered for scan. However, if these cells are used to hold off sets and resets of other cells so that another cell can be scannable, you must use this command to make them non-scan.

Add_nonscan_instances may invalidate data from other analysis stages. This command should be used before any X-bounding analysis, wrapper analysis and/or scan chain analysis. If you have already completed any of these steps when you issue the add_nonscan_instances command, it will result in an error. You can use the -force switch to override it, in which case the results of X-bounding analysis, wrapper analysis and scan chain analysis will be reset and you will have to repeat them.

Arguments

- **-INStances** *instance_spec*

An optional switch and value pair that causes the tool to ignore the specified instances during scan insertion. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also flagged as non-scan.

- **-Modules** *module_spec*

An optional switch and value pair that causes the tool to ignore all sequential cells within the instances of the specified modules during scan insertion. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tesson library model.

- **-scan_elements** *scan_element_spec*

An optional switch and value pair that causes the tool to explicitly ignore the specified scan elements during scan insertion. *scan_element_spec* can be a Tcl list of one or more scan elements or a collection of one or more scan elements.

- **-Control_signals** *pin_spec*

An optional switch and value pair that causes the tool to ignore the instances controlled by the specific control signals. *pin_spec* can be a Tcl list of one or more pins or a collection of one or more pins. You can only use the **-Control_signal** option in analysis mode.

- **-rtl_reg** *net_spec*

Note

 This switch is only available in dft -rtl context.

An optional switch and value pair that specifies a Tcl list of one or more net names or a collection of one or more net objects. During quick synthesis, if the net ends up being synthesized to a flip-flop, the net will end up being directly connected to the Q pin of a DFF cell. The `is_non_scannable` attribute value will be automatically be transferred to the DFF cell instance. The `is_non_scannable_reason` attribute for these DFT cell instances will return `user_specified_from_net`. This switch is only available in the dft -rtl context (with no sub-contexts). You use this switch when you want to declare an RTL register to be treated as non-scannable during DRC. You cannot specify the `add_nonscan_instance` command on the DFF cell instances directly because they do not exist as yet before the quick synthesis has happened when going from setup to analysis mode.

- **-All**

An optional switch that marks every scan cell in the design as a non-scan cell. When you issue the command with the **-All** switch, the tool completely clears the list of targeted cells, and you can then manually select an arbitrary subset of target cells using the `add_scan_instances` command.

- **-force**

An optional switch that directs the tool to forcefully issue the add_nonscan_instances command. In order to avoid using invalidated data, the tool will reset the results of any X-bounding analysis and wrapper analysis and delete all the scan modes and scan chain families. It is recommended that you re-run X-bounding and wrapper analysis and set the scan modes and scan chain families as needed after you issue the add_nonscan_instances command.

Examples

Example 1

The following example specifies that the tool ignore the sequential i_1006 instance when identifying and inserting the required scan circuitry:

```
add_nonscan_instances -instances i_1006
```

Example 2

The following example clears all previously identified scan cells and then adds two instances to the list of scan cells to be stitched:

```
add_nonscan_instances -all
add_scan_instances -instances i_1006 i_1007
insert_test_logic
```

Related Topics

[add_scan_instances](#)
[delete_nonscan_instances](#)
[insert_test_logic](#)
[report_seq_transparent_procedures](#)

add_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Prevents test logic insertion in the specified regions.

Usage

```
add_notest_points [-Control_only] [-OBserve_only]
{pin.pathname... | instance.pathname... | instance_expression} | -Pathfilename
```

Description

Prevents test logic insertion in the specified regions.

The add_notest_points command is used to mark specific regions of the design to be excluded when inserting test logic (control points, observe points, fixing bidirectionals, set/reset, x-bounding and clock lines). This command is maintained to provide backward compatibility with previous versions of the tool, but the no_control_point and no_observe_point attributes actually provide a more flexible mechanism for accomplishing this task.

Internally, the add_notest_points command sets the no_control_point and no_observe_point attributes to True for the specified pin_pathname, instance_pathname, or instance_expression. When an instance is specified, the attribute is set only for that specific instance and not for any child instances since this is automatically covered by inheritance.

If you specify a path file that contains delay paths, the tool marks each gate pin in the path by setting the no_control_point and no_observe_point attributes on each gate pin. This prevents control and observation points from being added to a critical path and thus prevents increasing the load on any of these gates in that path. The format of the file is the same as the file loaded with the read_fault_sites command. For more information on the format, refer to “[The Path Definition File](#)” in the *Tessent Scan and ATPG User’s Manual*.

The [report_notest_points](#) command displays all the pins in this list or can list the defined critical paths.

Arguments

- **-Control_only**
An optional switch that specifies that no control points will be added in the specified regions.
- **-OBserve_only**
An optional switch that specifies that no observe points will be added in the specified regions.

Note



By default, this command prevents the insertion of both control and observe points.

- ***pin.pathname***

A required, repeatable string that lists the pins that you do not want to use for insertion of controllability and observability.

- ***instance.pathname***

A repeatable string that lists the instances whose pins you do not want to use for controllability and observability insertion. All pins within that (hierarchical) instance are added to the list of pins that should be excluded from consideration.

- ***instance_expression***

A string representing a list of instances within the design. The string *instance_expression* is defined as:

```
{ string | string * } ...
```

The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

This argument does not support pathnames to objects below the instance level of a Tessent Cell library model. You can use a pathname expression to select several instances and the tool will then add_notest_points for all the pins on those instances; but if the expression specifies a location below the instance level of a Tessent Cell library model, the tool will issue an error message.

- **-Path *filename* (dft -test_points context only)**

A switch and filename pair that specifies the pathname to a file that contains critical path information. For more information on the format of the file, refer to “[The Path Definition File](#)” in the *Tessent Scan and ATPG User’s Manual*.

Examples

The following example specifies output pins that the tool cannot use for testability insertion:

```
add_notest_points i_1006/o i_1008/o i_1009/o
```

Related Topics

[delete_notest_points](#)

[set_test_point_insertion_options](#)

[report_notest_points](#)

add_observe_points

Context: dft -test_points

Mode: setup, analysis

Specifies user-defined observe points during test point insertion.

Usage

```
add_observe_points -location pin/port_spec
    [-clock pin/port_spec]
    [-enable pin/port_spec]
```

Description

Specifies user-defined observe points during test point insertion.

Arguments

- **-location** *pin/port_spec*

A required string that specifies the pathname to the observe point location where **pin/port_spec** can be a Tcl list of one or more pins/ports or a collection of one or more pin/port objects.

- **-clock** *pin/port_spec*

An optional switch and string pair that specifies the path name of the clock signal used to clock the observe point. This can be a collection or list, but should have only one element.

- **-enable** *pin/port_spec*

An optional switch and string pair that specifies the path name of an existing net used to enable/disable the observe point. This can be a collection or list, but should have only one element.

If you want to specify the default enable signal for the automatically identified observe points instead of manually adding each enable signal, use the [set_test_point_insertion_options](#) command as follows:

```
set_test_point_insertion_options -observe_points_enable
```

Examples

Example 1

The following example adds two observe points, and specifies the new scan cell model used to observe the inserted test points:

```
add_observe_points -location ss1/ix348/Z
add_observe_points -location ss1/ix302/Z
```

Example 2

The following example specifies the path names of existing nets that are to be used to enable/disable the control points and observe points respectively. Then it adds a control point and an observe point, and proceeds to insert the test points into the netlist:

```
set_test_point_insertion_options -control_point_enable cp_enable \
    -observe_point_enable op_enable
add_control_points -location jtc/i1216/OUT -type and
add_observe_points -location jtc/i1645/OUT
...
set_system_mode analysis
...
insert_test_logic
```

Related Topics

[add_control_points](#)
[report_test_points](#)
[delete_test_points](#)
[set_test_point_insertion_options](#)

add_output_masks

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Ignores any fault effects that propagate to the primary output pins you name.

Usage

`add_output_masks primary_output... | -All`

Description

patterns -scan Details

Ignores any fault effects that propagate to the primary output pins you name.

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command allows you the ability to flag primary output pins that do not have strobe capability. The tool classifies the faults whose effects only propagate to that observation point as ATPG_untestable (AU).

dft -scan Details

The tool uses primary output pins as the observe points during the scan identification process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the identification process.

There are specific design practices that mask specific primary output pins in order to group the sequential elements associated with those primary outputs (for example, if you are using wrapper chains). For more information on wrapper chains, refer to “[About Wrapper Chains](#)” in the *Tessent Scan and ATPG User’s Manual*.

You can set a default mask for all output and bidirectional pins using the [set_output_masks](#) command. You can add a hold value to a default mask with the `add_output_masks` command, or remove a hold value using the [delete_output_masks](#) command. To turn off the default masks for all output pins, you must use the [set_output_masks](#) command with the Off literal.

When you use the `add_output_masks` command in the `dft -scan` context, the wrapper analysis for that pin is excluded.

Note

 The pin constraints added when you use this command can be deleted using the [delete_output_masks](#) command. Pin constraints added in setup mode cannot be deleted in analysis mode and vice versa.

Arguments

- ***primary_output***
A repeatable string that specifies the name of the primary output pin you want to mask.
- **-All**
A switch that specifies to mask all primary output pins.

Examples

The following example specifies the primary output pins that will not have the strobe capability on the hardware tester:

```
add_output_masks qb1 qb2 qb3
```

Related Topics

[analyze_restrictions](#)

[delete_output_masks](#)

[report_output_masks](#)

[set_output_masks](#)

add_primary_inputs

Context: all contexts

Mode: setup

Adds a primary input (PI) also known as an input pseudo_port to the specified pins or nets.

Usage

```
add_primary_inputs pin_or_net_pathname... [-pseudo_port_name user_name]  
[-silent]
```

Description

Adds an input pseudo_port to the specified nets or pins. This command also designates the pseudo_port as user-class primary inputs (PI) rather than system-class PIs described in the original netlist.

When you issue the command, the tool disconnects the original drivers of the net such that the added PI becomes the only driver of the pin or net. The tool omits these PIs from the top-level interface when saving patterns.

Note

 The tool does not fault PIs created with this command. This allows interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

When you use the [write_patterns](#) -Mode_internal switch when saving serial or parallel simulation patterns, the test bench still forces the internal nodes on *pin_or_net_pathname* even though the corresponding PIs are not included in the top-level interface. When you save the patterns using the -Mode_external switch, the test bench will *not* force those internal nodes. In this case, unless the circuit generates values on the internal nodes corresponding to what ATPG assumed was forced on the corresponding PIs, you may get simulation mismatches.

Use the -pseudo_port_name switch to specify a list of internal pin pathnames to connect to a single, new primary input pin. You can display the user-class, system-class, or full-class PIs by using the [report_primary_inputs](#) command.

Arguments

- ***pin_or_net_pathname***

A required, repeatable string that specifies the pathname of a pin or net to which you want to add a primary input. The *pin_or_net_pathname* may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- **-pseudo_port_name *user_name***

An optional switch and string pair that specifies the name for a new primary input pin name that drives all the internal pins specified with the *pin_or_net_pathname* argument. The

user_name must be a simple name that begins with an alpha character and can contain only alphanumeric characters and underscore characters ('_'). Wildcards are not allowed.

- -silent

An optional switch that suppresses the transcription of notes the tool issues.

Examples

Example 1

The following example adds two new primary inputs to the circuit and places them in the user class of primary inputs:

```
add_primary_inputs indata2 indata4
```

Example 2

The following example adds a PI to the net originally driven by the output of a NAND gate, disconnecting the output of the NAND from the net. This happens within the tool only; the tool does not modify the original source netlist. The example then reports on the new PI and the original driver of the net. The example begins in setup mode. Notice that the tool reassigns gate IDs in the modified internal netlist.

```
set_gate_level primitive
create_flat_model
report_gates /ix157/Y

//  /ix157 (94)  NAND
//    A0      I  25-/reg_0_/Q
//    A1      I  33-/reg_1_/Q
//    Y       O  142-/d_2_mult_0_/A1

add_primary_inputs /d_2_mult_0_/A1
report_gates /ix157/Y

//  /ix157 (95)  NAND
//    A0      I  26-/reg_0_/Q
//    A1      I  34-/reg_1_/Q
//    Y       O
```

Example 3

The following example connects internal nets “xyz/abc/pina xyz/def/pina xyz/ghi” to a single, new primary input and disconnects the original drivers of the nets, making the added primary input the only driver of the nets. This is the default behavior. The name of the new primary input pin is “group_pi”.

```
add_primary_inputs xyz/abc/pina xyz/def/pina xyz/ghi -pseudo_port_name group_pi
```

Related Topics

[add_clocks](#)

[add_primary_outputs](#)

[delete_primary_inputs](#)
[report_primary_inputs](#)

add_primary_outputs

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Adds primary outputs to the specified net.

Usage

```
add_primary_outputs pin_or_net_pathname... [-pseudo_port_name user_name]  
[-silent]
```

Description

Adds primary outputs (PO) to the specified net.

The add_primary_outputs command adds an additional primary output to each specified pin or net. Once added, the tool defines them as user-class primary outputs. These primary outputs are not added to the top-level interface when saving mode_external patterns, such as STIL and WGL.

Note

 The tool does not fault POs created with the add_primary_outputs command. This allows interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

The tool defines the primary outputs described in the original netlist as system class primary outputs. You can display the user class, system class, or full classes of primary outputs using the report_primary_outputs command.

Arguments

- ***pin_or_net_pathname***

A required, repeatable string that specifies the pin or net to which you want to add primary outputs. The *pin_or_net_pathname* may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- **-pseudo_port_name *user_name***

An optional switch and string pair that specifies the name for a new primary output pin name that drives all the internal pins specified with the *pin_or_net_pathname* argument. The *user_name* must be a simple name that begins with an alpha character and can contain only alphanumeric characters and underscore characters ('_'). Wildcards are not allowed.

- **-silent**

An optional switch that suppresses the transcription of notes the tool issues.

Examples

The following example adds a new primary output to the circuit and places it in the user class of primary outputs:

```
add_primary_outputs outdata1
```

Related Topics

[add_primary_inputs](#)

[delete_primary_outputs](#)

[report_primary_outputs](#)

add_processors

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion (dft -edt context only)

Enables the tool to run multiple processors in parallel on multiple machines to reduce runtime.

Usage

```
add_processors {hostname[:{cpus | MAXcpu}]}...
```

Description

The add_processors command enables the tool to use multiple processors simultaneously on specified host machines when executing multiprocessing commands for ATPG or fault simulation. Slave processes can be created on the master host and on other hosts. Each process, whether the master or slave, can optionally use multiple processors via multithreading.

For more information, refer to “[Multiprocessing for ATPG and Simulation](#)” in the *Tessent Scan and ATPG User’s Manual*.

Running multiple processes in parallel can sometimes reduce runtime significantly, especially for large designs. Be sure to load your design in the tool before using the add_processors command. This allows the tool to more accurately estimate memory requirements for the added processes.

Normally, the tool executes as a single master process. When you make additional processors available with the add_processors command, while there is still a single master process, the tool starts additional slave processes and/or adds additional processors (threads) to existing processes for parallel execution and coordinates communication between all processors.

Multiprocessing commands that use slave processes alert you with a message when the design is being sent to slave processors. This message may not appear during subsequent commands if the slaves have already received the design and you have issued no commands in the interim that would affect the information received. There is currently a limit of 1024 processors per add_processors command.

Tip

 To stop the command from waiting any longer for requested slave processes not yet received, and proceed with those it has, use the Control-C key sequence.

Arguments

- **hostname**

A required, repeatable string or literal that specifies a host machine or job scheduler that the tool can use for slave processes.

The hostname choices are as follows:

- machine_name*** — A string that specifies a host machine by its network name.
- LOCALHOST** — A literal that specifies to use the machine you invoked the tool on as the host machine.
- IPaddress*** — A string that specifies a host machine by its IP address.
- LSF** — A literal that specifies to use available Load Sharing Function (LSF) host machines.
- SGE** — A literal that specifies to use available Sun Grid Engine (SGE) host machines.
- GENERIC** — A literal that specifies to use a host machine as determined by a custom job scheduler. This option requires that you first issue a [set_multiprocessing_options](#) command with the generic_scheduler argument to inform the tool how to launch the custom job scheduler.

Note

 When you add SGE or LSF slave hosts with the `add_processors` command, the tool estimates the available host memory and swap space needed for the slaves based upon the current memory usage of the tool and the number of requested processors. The tool includes these requirements in its requests for processors from the grid system. For this reason, it is best that you specify any setup commands that may have significant effects on memory usage (such as leaving setup mode and reading SDC and SDF files) prior to using the `add_processors` command. In the Tessent Shell tool, the design should be loaded before adding slave processors.

Note

 The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. See “[Multiprocessing Requirements](#)” in the *Tessent Scan and ATPG User’s Manual* for more information.

- `:{cpus | MAXcpu}`

A colon (:) followed by a positive integer or the MAXcpu keyword with which you can optionally specify the number of processors to use on the specified host machine or via a job scheduler. You can specify up to 32 processors per host. If you specify 0 processors, the tool issues a warning.

If you specify a positive integer, the tool adds that number of processors.

If using a job scheduler, the tool uses as many processors as the scheduler provides in the time allotted. The time allotted is determined by the setting of the scheduler_timeout variable, which you can change using the [set_multiprocessing_options](#) command.

If you specify the Maxcpu literal, the tool automatically determines the number of physical processors on the specified host and adds enough additional processors to bring the total number of running processors on that host up to the number of physical processors. The invocation default for SGE, LSF and generic grid schedulers is four processors; for the others it is one.

Note

 The Maxcpu literal does not apply to grid scheduling or generic scheduling. The tool's Maxcpu calculations are based on currently running processors without considering other slave processor specifications on the same command line. If, for example, you use the Maxcpu literal twice in the same command for a given host, the number of slave processors on that machine could exceed the number of available processors.

Examples

Example 1

The following example adds processors to the master host, which increases the thread count instead of adding additional slaves (unless multithreading has been disabled with the `set_multiprocessing_options` command):

```
add_processors localhost:4 ;
# adds 4 threads to the master process for a total thread count of 5

// Adding 4 threads to birdeye (master)
// Master with 5 threads running.

report_processors

// hosts           threads   arch      CPU(s)    %idle   free RAM  process size
// -----  -----  -----  -----  -----  -----
// birdeye (master)      5  x86-64   8 x 2.9 GHz   89%  113.91 MB  232.56 MB
// master with 5 threads running.
```

Example 2

The following example starts a distributed slave process on the remote host “odin” that runs with 4 total threads. The `add_processors` command normally starts only one process per slave host, and adds additional processors (to master or slave) as additional threads. (Exception: If multithreading is disabled with the `set_multiprocessing_options` command, `add_processors` creates multiple single-threaded slaves.)

```
add_processors odin:4

// Adding 4 threads to odin (new slave)
// Master with 1 thread and 1 slave with 4 threads running (5 total threads).
```

Related Topics

[compress_patterns](#)
[create_patterns](#)
[delete_processors](#)
[order_patterns](#)
[report_processors](#)
[set_multiprocessing_options](#)

add_read_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Specifies the off-state value for specified RAM read control lines.

Usage

```
add_read_controls {0 | 1} pin_pathname...
```

Description

Specifies the off-state value for specified RAM read control lines.

The add_read_controls command defines the circuit read control lines and assigns their off-state values. The off-state value of the pins or ports that you specify must be sufficient to keep the RAM outputs stable. The following are considerations for using this command:

- When the read control is already added as a clock, there is no need to also use add_read_controls for that port. This is the typical case, which means that for most usage it is unnecessary to explicitly use the add_read_controls command.
- If you want to have NR (non-clock) timing for read control lines that drive level-sensitive RAMs, you must add the read control using the add_read_control command; otherwise, the read control cannot pass DRC. NR timing for read control is not common.
- If a port drives the clock port of an edge-triggered RAM, the tool automatically infers that the port is a clock and adds it to the clock list. This is regardless of whether you added the port as a read control or not. For clocks that the tool infers, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0 | 1**

A required literal that specifies 0 or 1 as the off-state value for the RAM read control lines.

- ***pin_pathname***

A required, repeatable string that specifies the pins or ports you want to designate as RAM read control lines, and to which you are assigning the given off-state value.

Examples

The following example assigns an off-state value of 0 to two read control lines, r1 and r2:

```
add_read_controls 0 r1 r2
set_system_mode analysis
create_patterns
```

Related Topics

[analyze_control_signals](#)
[delete_read_controls](#)
[report_read_controls](#)

add_register_value

Context: all contexts

Mode: setup

Creates a static or dynamic variable by allowing you to specify a register identifier, *value_name*, and, optionally, the state string, *value_string*.

Usage

```
add_register_value value_name {[value_string [-Dynamic]
[-Radix {Binary | Decimal | Octal | Hexadecimal}] {-Width [integer]}
{-Pattern_count {SCan_test | CHain_test | ALI}}}
[-LSB_shifted_first] [-INput_pin_inversion] [-OUtput_pin_inversion]
```

Description

Creates a static or dynamic variable by allowing you to specify a register identifier, *value_name*, and, optionally, the state string, *value_string*.

When you specify a *value_string*, the variable is static by default unless you use the -Dynamic switch.

See “[Serial Register Load and Unload for LogicBIST and ATPG](#)” in the *Tessent Shell User’s Manual* for complete information.

Arguments

- ***value_name***
A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.
- ***value_string***
An optional state string whose default radix is binary. You can specify Z or X values in addition to numerical values appropriate for the radix. When you specify a *value_string*, the variable is static by default unless you use the -Dynamic switch. Note that you aren’t allowed to specify a *value_string* when you use the -Pattern_count switch.
- **-Dynamic**
An optional switch that forces the variable to be dynamic when you provide an optional *value_string*. By default, when you provide a *value_string*, the variable is static.
- **-Radix {Binary | Decimal | Octal | Hexadecimal}**
An optional switch and literal that specifies the radix of the *value_string*. If you specify the -Radix switch, then you must specify the register width using the -Width switch.

- -Width *integer*

A switch and integer pair that specifies the register width. When defining a dynamic variable, you must use this switch. If you do not know the actual width value until after test generation completes, then use the -Width switch without *integer*, and specify the width using the [set register value](#) command.

The maximum width of a register value is 1,000,000,000. A specified width larger than this produces an error.

- -Pattern_count {SCan_test | CHain_test | ALL}

A switch and literal pair that specifies using the total number of patterns generated by the tool when the variable is dynamic. Note that when you use this switch, you cannot specify a *value_string*. Choose from one of the following:

SCan test — Scan patterns.

CHain test — Chain patterns.

ALL — Both scan and chain patterns.

- -LSB shifted first

An optional switch that specifies shifting the less significant bit first into the register.

- -INput pin inversion

An optional switch that specifies an inversion on the input pin when loading the register.

- -OUtput pin inversion

An optional switch that specifies an inversion on the output pin when unloading the register.

Examples

The following example deletes all register value variables and then specifies two dynamic register value variables with a decimal radix and register width of 37.

Related Topics

delete register value

report register value

set register value

[add_rtl_to_gates_mapping](#)

Context: all contexts

Mode: setup

Adds name-mapping rules for RTL to post-synthesis/post-layout name mapping.

Usage

```
add_rtl_to_gates_mapping -register_name_map register_name_map  
| -instance_name_map_list string_pair_list
```

Description

This command allows you to add extra name-mapping rules for RTL to post-synthesis/post-layout name mapping.

During the flow from RTL to the final gate-level netlist, the tool stores a number of files, for example, TCDs (Tessent Core Description), ICL files, and so forth. These files contain design instance path names and RTL register instance path names. During synthesis and layout, the RTL register names are mapped to flip-flop instances, and the design hierarchy may be altered through grouping and/or ungrouping of various hierarchy levels. To handle the renaming, the tool performs name mapping to find the objects in the netlist using the names they had in the RTL. By default, the tool is configured to handle most common name-mapping cases. If the default name-mapping rules are not sufficient, you must provide additional name-mapping rules using the `add_rtl_to_gates_mapping` command. You typically do not need to adjust the mapping rules unless you used “ungroup -simple_names” during synthesis as shown in the example below.

You use the following commands and switches to return a collection of pins or instances that use the RTL post-synthesis/post-layout name mapping:

- `get_instances -match_rtl_reg`
- `get_instances -use_path_matching_options`
- `get_pins -use_path_matching_options`

Arguments

- **-register_name_map register_name_map**

A switch and string pair that specifies the RTL register renaming rule to add.

The register renaming rules use the following predefined variables:

- `%name` — This variable represents the original RTL register name without the index when it is a bus.
- `%(d0), %(d1)... %(d9)` — These variables represent register array indexes.

The following rules are provided by default and represent the default behavior of synthesis tools. You may need to add mappings if you have specified nonstandard naming rules in your synthesis tool.

```
"\% (name)_reg "
"\% (name)_reg_%(d0) "
"\% (name)_reg[%(d0)] "
"\% (name)_reg[%(d0)][%(d1)] "
"\% (name)_reg[%(d0)][%(d1)][%(d2)] "
"\% (name)_reg[%(d0)][%(d1)][%(d2)][%(d3)] "
"%(name)_reg_%(d0)"
"%(name)_reg_%(d0) "
"%(name)_reg_%(d0)_%(d1) "
"%(name)_reg_%(d0)_%(d1)_%(d2) "
"%(name)_reg_%(d0)_%(d1)_%(d2)_%(d3) "
"rtlcreg_(name)_%(d0)"
```

- **-instance_name_map_list string_pair_list**

A switch and string pair list that specifies instance name-mapping rules. By default, you do not need to specify mapping because the tool already maps constructs such as “corea_i1/coreb_i1” to “\corea_i1/coreb_i1” and “corea_i1_coreb_i1”. Even a hierarchy path that is part of a generated block such as “b1.b2[3].u2” is automatically mapped to “\b1.b2[3].u2” and “b1_b2_3_u2”. If you used “ungroup -simple_name” to ungroup coreb_i1 into corea_i1, you will need to specify a mapping from “corea_i1/coreb_i2” to “corea_i2” as shown in the example below.

Examples

The following example uses the `add_rtl_to_gates_mapping` command to specify that objects within instance coreb_i1 were ungrouped into their parent instance corea_i1 using the “ungroup -simple_name” command. The `report_rtl_to_gates_mapping` command invocation shows the added mapping. As described above, you typically never need to add mapping unless you have used the “ungroup -simple_name” command with your synthesis tool.

```
add_rtl_to_gates_mapping -instance_name_map_list {corea_i1/coreb_i1 corea_i1}
report_rtl_to_gates_mapping
```

```
// Index Register name map
// ----- -----
// 1  '\% (name) _reg '
// 2  '\% (name) _reg_%(d0) '
// 3  '\% (name) _reg[%(d0)] '
// 4  '\% (name) _reg[%(d0)][%(d1)] '
// 5  '\% (name) _reg[%(d0)][%(d1)][%(d2)] '
// 6  '\% (name) _reg[%(d0)][%(d1)][%(d2)][%(d3)] '
// 7  '%(name) _reg_%(d0)'
// 8  '%(name) _reg_%(d0)_'
// 9  '%(name) _reg_%(d0)_%(d1)_'
// 10 '%(name) _reg_%(d0)_%(d1)_%(d2)_'
// 11 'rtlcreg_%(name)_%(d0)'

// Index Original instance name Mapped instance name
// ----- -----
// 1      corea_i1/coreb_i1           corea_i1
```

Related Topics

[get_instances](#)
[get_pins](#)
[report_rtl_to_gates_mapping](#)

add_scan_chains

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup

Prerequisites: You must define the scan chain group with the add_scan_groups command prior to using this command.

Specifies a name for a pre-existing scan chain within the design.

Usage

```
add_scan_chains [-Internal | -Load_only] {chain_name group_name input_pin output_pin}...
```

For “dft -edt” context

```
add_scan_chains [-Internal] {chain_name group_name input_pin output_pin}...
```

Description

Specifies a name for a pre-existing scan chain within the design.

The add_scan_chains command defines a scan chain that exists in the design. A scan chain references the name of a scan chain group, which you must define prior to issuing this command.

A pre-existing scan chain is a serially connected set of scan cells that has been stitched together in a previous scan insertion operation, or that you created manually. In order for the tool to be able to correctly insert scan elements in the remainder of the design or to run rules checking on existing scan circuitry, you need to notify the tool of this circuitry, so it can treat those elements in the chain differently than the elements in the rest of the design.

When you use this command to notify the tool of a pre-existing scan chain, “dft -scan” context removes the scan cells in the scan chain from the eligible scan elements list. Because the tool removes the scan cells in the scan chain from the scan candidate list, the rules checker does not perform the usual scannability checks on those scan cells. However, the rules checker does perform additional rules checking based on the declaration of pre-existing scan elements that pertain to the checking of the validity and the operation of that scan chain.

If the design does have a pre-existing scan chain that you declare with the add_scan_chains command, you need to provide information on the operation of that scan chain in a test procedure file.

Arguments

- ***chain_name***

A required repeatable string that specifies the name of the scan chain to add to the specified scan group.

- ***group_name***

A required repeatable string that specifies the name of the scan chain group to add the specified scan chain to.

- ***input_pin***

A required repeatable string that specifies the input pin of the specified scan chain.

- ***output_pin***

A required repeatable string that specifies the output pin of the specified scan chain.

- -Internal

An optional switch that defines internal scan chains.

Note


The -Internal switch specifies compressed scan chains when using Tessent TestKompress with EDT logic. Do not use it to define uncompressed scan chains (chains connected directly to a primary input and primary output and not driven by or observed through the Tessent TestKompress logic). See the “[Scan Chain Insertion](#)” section in the *Tessent TestKompress User’s Manual*.

- -Load_only (Not supported in the hybrid TK/LBIST flow)

An optional switch that specifies the definition of a load-only scan chain which is an uncompressed scan chain in which the scan input is a primary input and the scan output is an internal pin that is only used for the purpose of scan chain tracing. A load_only scan chain does not need to be unloaded which saves chip-level pins and scan data. You can specify a load_only scan chain for special cases such as on-chip clock controllers that only need to be loaded but do not need to be unloaded.

The tool automatically adds OX constraints on all cells of the chain using the chain masking mechanism, since those cells will not be unloaded and observed. In the generated patterns or mode_external test bench, the internal scan chain outputs are ignored. These chains can be added with other normal scan chains, and/or with EDT blocks. However, the load-only chains must be uncompressed and loaded only from ports, not a decompressor.

Note


This switch does not support mapping EDT patterns to single bypass chain mode.

Also, just like normal uncompressed chains, load-only uncompressed chains should not be added during the EDT IP phase.

Examples

Example 1

The following example defines two scan chains (chain1 and chain2) that belong to the scan group (grp1):

```
add_scan_groups grp1 atpg.testproc
add_scan_chains chain1 grp1 si1 so1
add_scan_chains chain2 grp1 si2 so2
```

Example 2

The following example shows add_scan_chains commands similar to those Tessent TestKompress would write in the *_edt.dofile dofile, during the IP Creation Phase, for the two scan chains of the preceding example. By default, the top level module name “/my_design” is prefixed to the pin names.

```
add_scan_groups grp1 *_edt.testproc
add_scan_chains -internal chain1 grp1 /my_design/si1 /my_design/so1
add_scan_chains -internal chain2 grp1 /my_design/si2 /my_design/so2
```

Related Topics

[add_scan_groups](#)

[delete_scan_chains](#)

[report_scan_chains](#)

[add_scan_groups](#)

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup

Specifies a group name for a set of pre-existing scan chains in a design.

Usage

`add_scan_groups {group_name test_procedure_filename}...`

Description

Specifies a group name for a set of pre-existing scan chains in a design. Only one group name can be specified per session.

If the design has pre-existing scan chains, those scan chains must belong to a scan group. The `add_scan_groups` command also specifies the corresponding test procedure file for that scan group. A sequence of procedures in the test procedure file defines the operation of the scan chains within a scan group. The scan group must have a corresponding test procedure file.

When needed, you can also use the `add_scan_groups` command to provide initialization values for non-scan memory elements. If the design does not have pre-existing scan chains and you need to initialize some non-scan memory elements for scannability checking, you can specify “dummy” as the `group_name` along with a `test_procedure_filename`. Then, within the test procedure file, you can use the optional `test_setup` procedure to initialize the non-scan memory elements. For complete information, see “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

As a result, in `dft -scan` context, the tool models the non-scan memory elements as a TIE1 or TIE0.

Note

 You must also define these same elements as non-scan using the [add_nonscan_instances](#) command.

Arguments

- **`group_name`**

A required string that specifies the name of the scan chain group that you want to add to the system.

- **`test_procedure_filename`**

A required string that specifies the name of the test procedure file that contains the information for controlling the scan chains in the specified scan chain group.

Examples

The following example defines a scan chain group, group1, which loads and unloads a set of scan chains, chain1 and chain2, by using the procedures in the file, *scanfile*:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 testout2
add_scan_chains chain2 group1 indata4 testout4
```

Related Topics

[add_scan_chains](#)

[delete_scan_groups](#)

[report_scan_groups](#)

[read_procfile](#)

[write_procfile](#)

add_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Adds specified instances to the scannable instance list.

Usage

`add_scan_instances -INStances instance_spec | -Modules module_spec`

Description

Adds specified instances to the scannable instance list.

When inserting scan chains, the add_scan_instances command ensures that the tool includes specific sequential instances in the scan list. The tool generates the scan list during the scan identification process. If the specified instance does not pass the rules checking process, it is not included in the identified scan list.

You can use the [delete_scan_instances](#) command to remove the exception status that you have attached to an instance with the add_scan_instances command. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

Note

 The add_scan_instances command can be used to override the default behavior for the S4 rule (when handling is set to Note), which forces cells with S4 violations to remain non-scan. Likewise, the add_scan_instances command can override the behavior of the S7 rule when its handling is set to Warning, which forces cells with S7 violations to remain non-scan. This provides a mechanism to override the S4 and/or S7 rule behavior for specific cells.

Arguments

- -Instances *instance_spec*

An optional switch and value pair that causes the tool to ignore the specified instances during scan insertion. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also flagged as non-scan.

- -Modules *module_spec*

An optional switch and value pair that causes the tool to ignore all instances within the specified modules during scan insertion. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tesson library model.

Examples

The following example adds two sequential instances to the identified scan list and then runs the scan identification process.

```
add_scan_instances -instances i_1006 i_1007  
insert_test_logic -number 10
```

Related Topics

[delete_scan_instances](#)

[report_seq_transparent_procedures](#)

[add_scan_mode](#)

Context: dft -scan

Mode: analysis

Specifies a chain scan mode, which defines one or more scan chains.

Usage

add_scan_mode *name*

```

[-chain_length { integer | unlimited } | -chain_count { integer | unlimited }]
[-type { internal | external | unwrapped }]
[-include_chain_families obj_spec]
[-include_elements obj_spec]
[ { -enable_connections obj_spec [-enable_code binary_string | auto } | \
  -enable_dft_signal string ]
[-si_timing { leading_edge | trailing_edge | any_edge }]
[-so_timing { leading_edge | trailing_edge | any_edge }]
[ -edt_instances obj_spec | {-si_connections obj_spec -so_connections obj_spec } ]
[ -si_associated_ports obj_spec -so_associated_ports obj_spec ]
[-si_port_format format]
[-so_port_format format]
[-port_index_start_value integer]
[-port_scalar_index_modifier integer ]
[-single_class_chains { On | Off }]
[-single_clock_domain_chains { On | Off }]
[-single_clock_edge_chains { On | Off }]
[-single_power_domain_chains { On | Off }]
[-single_cluster_chains { On | Off }]
[-single_wrapper_type_chains { On | Off }]
[-lockup_cell_type { latch | flop }]
[-si_lockup_cell_type { latch | flop }]
[-so_lockup_cell_type { latch | flop }]
[-unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input }]
[-si_pipelining { wrapper_chains_only | all_chains | off }]
[-si_pipelining_type {non_scan_flop | holding_scan_flop}]
```

Description

Use this command to specify a chain scan mode, which defines one or more scan chains. Chain configurations specific to each scan mode are achieved by multiplexing scan path at certain inflection points computed by the scan insertion tool.

Arguments

- ***name***

A required string that uniquely describes the scan mode. When using DFT signals, if the specified scan mode name corresponds to the name of a DFT scan mode signal name, then

the command will implicitly infer both the type (-type) and the enable signal (-enable_dft_signal) for the mode; if the specified scan mode name does not match the name of a DFT scan mode signal, then you must specify the enable signal using the option “-enable_dft_signal” and the command will implicitly infer the mode type (-type) from it.

- **-chain_length { integer | unlimited }**

An optional integer that specifies the target length of scan chains for a particular mode. By default, the no limit (unlimited) is imposed on the chain length. The minimum scan chain length is 1.

- **-chain_count { integer | unlimited }**

An optional integer that specifies the number of scan chains that should be constructed for the specified mode. By default, the chain count is set to the number of connections specified with the -si/so_connections or by the number of pins of the EDT controller(s) specified with the -edt_instances option, otherwise if the latter options were not used, then no limit (unlimited) is imposed on the chain count.

- **-type internal | external | unwrapped**

An optional string and literal pair that specifies the type of scan mode. A mode of type external only considers wrapper scan elements in its default population, whereas internal and unwrapped modes consider all scan elements. For more information, see “[Unwrapped Cores Versus Wrapped Cores](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-include_chain_families *obj_spec***

An optional value that specifies a list of chain families to include in this mode – defaults to all chain families if no “-include_” option is specified.

- **-include_elements *obj_spec***

An optional value that specifies a list of scan elements to include in this mode – defaults to all scan elements if no “-include_” option is specified.

- **-enable_connections *obj_spec***

An optional value that specifies one or more top-level port(s) or existing pin(s) to enable this mode. By default, a scan_mode enable port with a name of the form “ts_stm%ds0” gets created, where %d corresponds to the mode’s numeric ID. If more than one connection is specified, the same set of connections will be required for each defined mode, each requiring a unique enable_code; the latter will be specified explicitly using “-enable_code <binary>” or implicitly assigned by the tool (-enable_code auto).

- **-enable_code { *binary_string* | auto }**

An optional value that specifies a binary number (for example, “00110”) to be explicitly used to decode this mode using the specified enable_connections. By default or when the value “auto” is supplied, the tool automatically assigns a unique code to decode this mode based on the total number of scan modes. Each mode must have a unique code that uses all of the specified connections, or an error will be generated.

- **-enable_dft_signal *string***

An optional value that specifies the name of a dft scan mode signal as defined with the command “get_dft_signal <name> -usage scan_mode” to enable this mode. The scan mode type associated with the dft signal is used if the –type option was not specified, or checked for compatibility otherwise.

- **-si_timing { leading_edge | trailing_edge | any_edge }**

An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in chain to control the capture edge of the first cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is “leading_edge”.

- **-so_timing { leading_edge | trailing_edge | any_edge }**

An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain for this scan mode. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in chain to control the change edge of the last cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is “trailing_edge”.

- **-edt_instances *obj_spec***

An optional value that specifies EDT instances to connect the scan chains to. When core descriptions for the specified instances are detected, then the tool automatically sets the -si/ so_connections to the instances scan pins.

When you use this option, the tool automatically infers the –chain_count value to be the number of available scan connection(s) on the specified EDT controller(s), and issues a message like this:

```
// command: add_scan_mode int_mode -type internal -si_pipelining off -edt_instance
// {test[0][0].my_white/point_scalar_mult_rtl_tessent_edt_lbist_c2_inst}
// Reading core description file ./tsdb_outdir/instruments/
point_scalar_mult_rtl_edt_lbist.instrument/
point_scalar_mult_rtl_tessent_edt_lbist_c2.tcd
// Setting the chain count to the number of specified scan connections (50).
// Automatically inferring '-enable_dft_signal int_mode' from the specified scan
// mode name.
```

This inferred mode chain_count now also accounts for the EDT scan connections from families included in the mode, as follows:

```
// Increasing the chain count to 140 in order to account for chain family EDT scan
// connections (90)
```

- **-si_connections *obj_spec* -so_connections *obj_spec***

Optional string and literal pairs that specify SI/SO connections to existing pins or ports to be used first when connecting newly allocated scan chains. The specified values can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). They may be existing top-level input port(s) for

-si_connections and existing top-level output port(s) for -so_connections, or alternatively -si_connections may correspond to existing output pin(s) and -so_connections to existing input pin(s). By default, -chain_count is implicitly set to the number of si/so_connections specified; if this is not the desired behavior, then you should also set an explicit -chain_count.

- -si_associated_ports *obj_spec* - so_associated_ports *obj_spec*

Optional string and literal pairs that specify existing top-level port(s) associated to the internal connections identified using -si/so_connections. These are used in the .tcd file to describe the scan ports of the chains at the core boundary.

- -si_so_port_format *format*

An optional string that specifies the port format. The default si_so_port_format is set to “ts_%s_si/so[%d]” when there are more than one scan mode, otherwise it defaults to “ts_si/so[%d]”. The mode placeholder “%s” gets replaced by the chain mode name, whereas the index placeholder “%d” gets substituted by the chain index. A precision number may be used when specifying the index placeholder, for example the format “lsi_si%2d” would cause port names such as “lsi_si00”, “lsi_si01” to be created. Note that leading zeros are inserted automatically when needed making “%2d” implicitly equivalent to “%02d”.

- -port_index_start_value *integer*

An optional integer that specifies the starting port index. The starting index is 0 by default, but can be overridden by this option.

- -port_scalar_index_modifier *integer*

An optional switch and integer that specifies the increment used between each chain port indices. The default is 1.

- -single_class_chains { on | off }

An optional switch and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. The default is off, so the tool will allow distributing wrapper and internal cells onto the same chains.

- -single_clock_domain_chains { on | off }

An optional switch and literal pair that restricts a scan chain to a single clock domain. The default is off.

- -single_clock_edge_chains { on | off }

An optional switch and literal pair that restricts a scan chain with flops having a single edge. The default is off.

- -single_power_domain_chains { on | off }

An optional switch and literal pair that restricts a scan chain to a single power domain using the attribute "power_domain_name". The default is off.

- `-single_cluster_chains { on | off }`
An optional switch and literal pair that restricts a scan chain to a single cluster using the attribute "cluster_name". The default is on.
- `-single_wrapper_type_chains { on | off }`
An optional switch and literal pair that specifies that “-single_class_chains ON” controls whether to mix input wrapper and output wrapper cells in the same chain. The default is off.
- `-lockup_cell_type { latch | flop }`
An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch.
- `-si_lockup_cell_type { latch | flop }`
An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch.
- `-so_lockup_cell_type { latch | flop }`
An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

Note

 By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

- `-unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input }`
An optional switch that selects how to handle unused EDT chain pins. Options are:
 - insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.
 - tie_compactor_input — ties-off the EDT compactor.
 - auto — selects “insert_pipelining_flop” for EDT controllers with bypass chains, and “tie_compactor_input” for all other EDT controllers. This is the default.
- `-si_pipelining_wrapper_chains_only | all_chains | off }`
An optional switch and string that controls whether a pipelining flop gets inserted at the beginning of each chain if the first scan element of the chain does not have the attribute “no_si_pipelining” set and is not an OCC segment. When set to all_chains, all scan chains get a pipelining flop. When set to wrapper_chains_only, only chains of internal or external mode types and composed of wrapper scan elements get a pipelining flop. The default is wrapper_chains_only.
- `-si_pipelining_type {non_scan_flop | holding_scan_flop}`
An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

add_scan_segments

Context: dft -scan, dft -test_points -no_rtl

Mode: setup

Adds one or more existing scan segment(s).

Usage

```
add_scan_segments scan_segment_name
  [ -no_trace ] [ -length integer ] [ -on_module obj_spec ]
  -si_connections obj_spec -so_connections obj_spec
  -clock_pins obj_spec [ -clock_update_edges { leading | trailing }... ]
  -scan_enable_pins obj_spec [ -scan_enable_inversion { false | true }... ]
  [ -set_pins obj_spec [ -set_inversion { false | true }... ] ]
  [ -reset_pins obj_spec [ -reset_inversion { false | true }... ] ]
  [ -ste_pins obj_spec [ -ste_inversion { false | true }... ] ]
```

Description

Use this command to add one or more existing scan segment(s).

Arguments

- ***scan_segment_name***
A required string that specifies the name of a scan segment.
- **-no_trace**
An optional switch that specifies to skip tracing even if a scan segment procedure file was read in; it can only be set if –length is specified. Defaults to OFF.
- **-length *integer***
An optional integer that specifies the length of the scan segment; a non-null value is required if -no_trace is used.
- **-on_module *obj_spec***
An optional switch and value pair that specifies an anchor module from which port specifications are relative to. Defaults to the top module.
- **-si/so_connections *obj_spec***
A required switch and value pair that specifies connections to existing SI/SO pins or ports. The specified value can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). The specified values corresponds to existing top-level input port(s) or output pin(s) for -si_connections and existing top-level output port(s) or input pin(s) for -so_connections.
- **-clock_pins *obj_spec***
A required switch and value pair that specifies a collection of list of pins. Once provided, the tool can perform S rule checks for the clocks from the scan segment boundary. The first and

the last elements of the collection correspond to the clocks for the first and last scan cells of a scan segment. The intermediate elements correspond to the clocks for the remaining scan cells in the scan segment. If there is just one element, then the tool assumes that the scan segment is using one clock.

- **-clock_update_edges { leading | trailing }...**

A repeatable switch and literal pair that specifies a list of clock edges at which the scan cells are capturing or the output is changing (for example, LE or TE). It is a list of edges. The first and the last elements of the list correspond to the edges for the first and the last scan cells in a sub-chain. There is no need to have more than two elements in the list. If there are more than two elements, the tool ignores them. If there is only one element, then the tool assumes the first and last scan cells to change at the same edge.

- **-scan_enable_pins *obj_spec***

A required switch and value pair that specifies a collection or list of scan_en pins driving the scan segment.

- **-scan_enable_inversion { false | true }**

An optional switch and literal pair that specifies whether there is inversion in the scan_en pin between the cell and the segment container. It is a list of boolean values.

- **-set_pins *obj_spec***

An optional switch and value pair that specifies a collection or list of set pins for the scan cells in an existing segment.

- **-set_inversion { false | true }**

An optional switch and literal pair that specifies whether it is an active high or low set pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins.

- **-reset_pins *obj_spec***

An optional switch and value pair that specifies a collection or list of reset pins for the scan cells in an existing segment.

- **-reset_inversion { false | true }**

An optional switch and literal pair that specifies whether it is an active high or low set pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins.

- **-ste_pins *obj_spec***

An optional switch and value pair that specifies a collection or list of scan test enable pins driving the scan segment.

- **-ste_inversion { false | true }**

An optional switch and literal pair that specifies whether it is an active high or active low scan test enable pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins. The default is false.

add_simdut_fault

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.

Usage

```
add_simdut_fault -signal signal_name -fault [0 | 1]
```

Description

SimDUT uses the Verilog simulation capabilities to force the signal to 0 or 1. The Verilog capability has the following behavior: When you specify a pin pathname for the fault you want to inject, Verilog forces the associated signal to the specified value.

When you inject a fault at a specific pin pathname, SimDUT actually injects the fault at the whole net, not only on the specific pin. The fault acts like a fault injected on the upstream driver pin of the associated net. If you specify the input port of a cell and this port is attached to a net with fanout, then the fault acts like a fault injected on the driver port of that net.

When you execute a test, SimDUT simulates the injected fault as if it were an actual defect in silicon, collects the simulated failing cycles, and returns the results.

You can execute this command multiple times. When you specify a particular signal more than once, the last command that specifies the signal overrides all previous specifications. For example, if you first add a stuck-at-0 fault to signal A and later add a stuck-at-1 to signal A, SimDUT only adds the stuck-at-1 fault to signal A.

Arguments

- **-signal** signal_name

A required switch and string pair that specifies the name of a signal from the Verilog netlist. Refer to “[SimDUT Signal Naming Convention](#)” in *Tessent SiliconInsight User’s Manual for Tessent Shell* for more information

- **-fault** [0 | 1]

A required switch and literal that specifies the fault type as follows:

0 - Specifies stuck-at fault type SA0.

1 - Specifies stuck-at fault type SA1.

Examples

The following example injects an SA0 fault into the SIMDUT_TB.DUT_inst.memory00.q_a signal.

```
add_simdut_fault -signal SIMDUT_TB.DUT_inst.memory00.q_a -fault 0
```

Related Topics

[delete_simdut_fault](#)

[report_simdut_faults](#)

[execute_cdp_test](#)

add_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Creates a new user-defined simulation context.

Usage

```
add_simulation_context context_name [-copy_from source_context_name] [-silent]
```

Description

Creates a new user-defined simulation context.

The add_simulation_context command allows you to create a new user-defined simulation context in which the tool can perform simulation and introspection. Initially, the tool propagates tied values, such as power and ground, and then sets the remaining simulation values to X. The command automatically sets the new simulation context to be the current simulation context.

Arguments

- ***context_name***

A required string that specifies the name of the context. The *context_name* has to be different than one of the predefined context names listed in [Table 6-7](#) on page 2023.

- **-copy_from *source_context_name***

An optional switch and string that creates a new simulation context by copying an existing simulation context. Using this switch, you can bypass the initialization work done by the tool and build up a new simulation context. See [Example 3](#).

- **-silent**

An optional switch that turns off the display of the message:

```
// 'sim_x' is the current simulation context.
```

Examples

Example 1

The following example shows how to create a new simulation context:

```
SETUP> add_simulation_context myContext
// 'myContext' is the current simulation context.

SETUP> get_simulation_context_list -user_defined {myContext}
```

Example 2

The following example shows creates a new simulation context by generating an empty simulation context and subsequently copying another simulation context:

```
add_simulation_context $context
copy_simulation_context -from $other_context
set_current_simulation_context $context
```

Example 3

The following example illustrates using the -copy_from switch to create a new simulation context by copying an existing simulation context:

```
add_simulation_context $context -copy_from $other_context
set_current_simulation_context $context
```

Related Topics

[add_simulation_forces](#)
[get_simulation_value_list](#)
[copy_simulation_context](#)
[report_simulation_contexts](#)
[delete_simulation_contexts](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[simulate_clock_pulses](#)
[get_current_simulation_context](#)
[simulate_forces](#)
[get_simulation_context_list](#)
[set_current_simulation_context](#)

add_simulation_forces

Context: dft, patterns

Mode: setup, analysis

Prerequisite: The flat model must already exist

Forces one or more *gate_pin* objects to the specified value.

Usage

`add_simulation_forces obj_spec -values values`

Description

Forces one or more *gate_pin* objects to the specified value(s).

The `add_simulation_forces` command forces the specified *gate_pin* object(s) to the specified force value(s) within the current simulation context. The actual simulation is not triggered until you issue the `simulate_forces` or `simulate_clock_pulses` command. If a specified *gate_pin* object has already been forced by an earlier `add_simulation_forces` command, and the tool has not yet simulated the previous force, the new force value overwrites the existing one in case of a conflicting value. Note that adding a simulation force also updates the *gate_pin* attribute named `simulation_value`.

This command is allowed for user-defined simulation contexts only.

Arguments

- ***obj_spec***

A required value that specifies one or more *gate_pin* objects.

- **-values *values***

A required switch and list of values, which specifies the force value applied to the specified *gate_pin* objects. *values* can be a Tcl list of values or a string of values.

The number of *values* must be one or must match the number of specified *gate_pin* objects. Valid values are: X, 0, 1, Z.

If only one value is specified, then the tool applies this value to all gates in the *gate_pin* collection.

Examples

The following example shows the effect of adding a force on a particular *gate_pin* object, and then triggering the tool to simulate that force:

```
ANALYSIS> set_current_simulation_context myContext  
ANALYSIS> add_simulation_forces [get_gate_pins l1] -value 1  
ANALYSIS> report_simulation_forces
```

Value	Object
1	I1

ANALYSIS> get_simulation_value_list [get_gate_pins I1]

{ x }

ANALYSIS> simulate_forces

ANALYSIS> get_simulation_value_list [get_gate_pins I1]

{ 1 }

Related Topics

[add_simulation_context](#)
[get_simulation_value_list](#)
[copy_simulation_context](#)
[report_simulation_contexts](#)
[delete_simulation_contexts](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[simulate_clock_pulses](#)
[get_simulation_context_list](#)
[simulate_forces](#)

add_synchronous_clock_group

Context: dft, dft -scan, patterns

Mode: setup

Defines a set of clocks that are synchronous (compatible) with each other.

Usage

`add_synchronous_clock_group {object_spec | -all}`

Note

 Synchronous clock groups have no impact on scan chain analysis, scan insertion, test point identification, and EDT IP insertion (in the dft -test_points and dft -edt sub-contexts).

Description

Defines a set of clocks that are synchronous (compatible) with each other.

The `add_synchronous_clock_group` command defines the specified interacting clocks as synchronous during capture cycles (that is, there is no clock skew between them). By specifying synchronous clocks explicitly, ATPG is allowed to pulse them at the same time to potentially reduce the generated test set size and improve test coverage.

You can use this command in combination with “`set_scan_insertion_options -enable_retiming skip_synchronous_clocks`” to avoid inserting lockup cells between synchronous clock groups.

Note

 Simulation mismatch errors are likely if you erroneously define clocks as synchronous that are not.

Arguments

- ***object_spec***
A Tcl list or collection of pin or net objects.
- **-all**
A switch that specifies all defined clocks.

Examples

The following example creates two clock groups from different sets of compatible clocks:

SETUP> report_clocks

User-defined Clocks (4) :

Sync and Async Source Clocks				
Name	Off	State	Constraints	Internal
I1	0			No
I2	0			No
I3	0			No
I4	0			No

SETUP> add_synchronous_clock_group {I1 I2}

SETUP> report_synchronous_clock_groups

User-defined Synchronous Clock Group

```
// No.  Clocks (Labels)
// --- -----
// 1    I1
//      I2
```

SETUP> add_synchronous_clock_group {I3 I4}

SETUP> report_synchronous_clock_groups

User-defined Synchronous Clock Groups

```
// No.  Clocks (Labels)
// --- -----
// 1    I1
//      I2
// 2    I3
//      I4
```

Related Topics

[add_clocks](#)
[delete_synchronous_clock_group](#)
[get_synchronous_clock_groups](#)
[report_clock_domains](#)
[report_synchronous_clock_groups](#)
[set_clock_restriction](#)

add_tied_signals

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Adds a value to floating signals or pins.

Usage

```
add_tied_signals {0 | 1 | X | Z} floating_object_name... [-Pin]
```

Description

Adds a value to floating signals or pins.

The add_tied_signals command assigns a specific value to not-clearly-defined floating signals or pins. If there are floating signals or pins in the design, a warning appears when you leave the setup mode. If you do not assign a specific value, the tool ties the signal or pin values to the default value. To change the default tied value, use the set_tied_signals command.

When you add_tied_signals or pins, the tool places them into the user class. When the netlist ties signals or pins to a value, the tool places them into the system class.

Note

 The tool will not tie a signal that is connected to bidirectional pins. For example, with a Verilog netlist that has a top level pin named Vdd of type “inout”, you cannot use “add_tied_signals 1 Vdd -pin” to tie Vdd high.

Arguments

- **0**
A literal that ties the floating nets or pins to logic 0 (low to ground).
- **1**
A literal that ties the floating nets or pins to logic 1 (high to voltage source).
- **X**
A literal that ties the floating nets or pins to unknown.
- **Z**
A literal that ties the floating nets or pins to high-impedance.
- ***floating_object_name***
A required, repeatable string that specifies the floating nets or pins to which you want to assign a specific value. The tool assigns the tied value to all floating nets or pins in all modules that have the names you specify.

For pin names, you must include the -Pin switch on the command line; otherwise, the tool assumes the name is a net name.

- -Pin

An optional switch specifying that the *floating_object_name* argument that you provide is a floating pin name.

Examples

The following example ties all floating signals in the circuit that have the net names vcc and vdd, to logic 1 (tied to high):

```
add_tied_signals 1 vcc vdd
```

Related Topics

[delete_tied_signals](#)

[report_tied_signals](#)

[set_tied_signals](#)

add_to_collection

Context: unspecified, all contexts

Mode: all modes

Adds objects to a base collection and returns the new collection without affecting the original base collection.

Usage

add_to_collection *base_collection* *object_spec* [-unique]

Description

Adds objects to a base collection and returns the new collection without affecting the original base collection.

By default, objects that exist in both the *base_collection* and the *object_spec* are duplicated in the resulting collection. You can use the *-unique* switch to remove duplicates.

Arguments

- ***base_collection***

A required value that specifies the base collection to which objects are to be added. This base collection is copied to a new collection, and objects matching *object_spec* are added to the new collection. The *base_collection* can be an empty collection (empty string).

- ***object_spec***

A required value that specifies a Tcl list of one or more object names, or a collection of one or more objects to add to the *base_collection*. The *object_spec* can be an empty collection (empty string). If *object_spec* is empty, the result is a copy of the *base_collection*.

- **-unique**

An optional switch that removes duplicate objects from the resulting collection. By default, duplicate objects are not removed.

Examples

Example 1

The following example adds two collections, *a* and *b*, to create a new collection called *sum*.

```
set a [get_ports a*]
set b [get_ports b*]
set sum [add_to_collection $a $b]
```

Example 2

The following example creates a collection of all nets connected to pins of design instances matching *u1/myPrefix**. Note that using the *-unique* option automatically removes duplicate nets.

```
set sum ""
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
    set inputs [get_pins -of_instance $inst -direction input]
    set outputs [get_pins -of_instance $inst -direction output]
    set sum [add_to_collection $sum [get_fanins $inputs -stop_on net] -unique]
    set sum [add_to_collection $sum [get_fanouts $outputs -stop_on net] -unique]
}
```

Note

 This example could have used the `append_to_collection` command instead, which would have been more efficient because in this example, the intermediate collections are destroyed as soon as they are added.

Related Topics

[append_to_collection](#)
[compare_collections](#)
[copy_collection](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

add_write_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Specifies the off-state value for specified RAM write control lines.

Usage

```
add_write_controls {0 | 1} pin.pathname...
```

Description

Specifies the off-state value for specified RAM write control lines.

The add_write_controls command defines the circuit write control lines and assigns their off-state values. The off-state value of the pins or ports that you specify must be sufficient to keep the RAM outputs stable. The following are considerations for using this command:

- When the write control is already added as a clock, there is no need to also use add_write_controls for that port. This is the typical case, which means that for most usage it is unnecessary to explicitly use the add_write_controls command.
- If you want to have NR (non-clock) timing for write control lines that drive level-sensitive RAMs, you must add the write control using the add_write_control command; otherwise, the write control cannot pass DRC. NR timing for write control is not common.
- If a port drives the clock port of an edge-triggered RAM, the tool automatically infers that the port is a clock and adds it to the clock list. This is regardless of whether you added the port as a read control or not. For clocks that the tool infers, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0 | 1**
A required literal that specifies 0 or 1 as the off-state value for the RAM write control lines.
- ***pin.pathname***
A required, repeatable string specifying the pins that are write control lines for the RAM and to which you want to assign an off-state value.

Examples

The following example assigns an off-state to two write control lines, w1 and w2:

```
add_write_controls 0 w1 w2
set_system_mode analysis
create_patterns
```

Related Topics

[analyze_control_signals](#)

[delete_write_controls](#)

[report_write_controls](#)

analyze_atpg_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.

Usage

```
analyze_atpg_constraints {-AUto | -All | [{pin_pathname | gate_id# | function_name}...  
[-Satisfy | -Exclusive]]} [-Bus]
```

Description

Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.

If you issue the `analyze_atpg_constraints` command without any arguments, the default is `-All`. When the command finishes, the tool displays a message that indicates whether the analysis passed, failed, or aborted the ATPG constraint analysis.

Note

 With EDT on, Tessent TestKompress support of this command is limited: it will not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

Arguments

The following lists the three methods for naming the objects for which you want to analyze the constraints. You can use any number of the three argument choices, in any order.

If you only specify an object name when you issue this command, by default, the tool performs the satisfiability (-Satisfy) analysis.

- **-Auto**

An optional switch that automatically tries to locate the atpg constraint that cannot be satisfied. The analysis checks to see if any single constraint cannot be satisfied, then reports each constraint that cannot be satisfied (given the current abort limit and other restrictions). Sometimes, each constraint can be satisfied by itself, but some set of constraints cannot all be satisfied. In this case, `-Auto` switch proceeds to a second analysis where it adds atpg constraints to a set to create a minimal set that can't be satisfied.

- **-All**

An optional switch that specifies for the tools to perform the ATPG analysis simultaneously for all the current ATPG constraints. If you do not specify an object name, this is the command default.

- ***pin_pathname***

A repeatable string that specifies the pathname to the pin on which you are analyzing the constraints.

- **gate_id#**

A repeatable integer that specifies the gate identification number of the gate on which you want to analyze the constraints.

- **function_name**

A repeatable string that specifies the name of a function you created with the `add_atpg_functions` command. If you generated the ATPG function with the `-Cell` option and added constraints with the `-Cell` option, then the tool also analyzes the constraints on all the cells affected by that ATPG function.

- **-Satisfy**

An optional switch that specifies for the ATPG process to attempt to create a pattern that satisfies the selected ATPG constraint. This is the default if you specify an object name without a switch. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the `-Bus` switch).

When the command finishes, the tools display a message that indicates whether the analysis passed (the ATPG process successfully generated a pattern), failed (the ATPG process could not find any possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis passes, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- **-Exclusive**

An optional switch that specifies for the ATPG process to attempt to create a pattern that sets the selected ATPG constraint at a value different from its constrained value. This test's intent is to ensure that such a pattern does not exist. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the `-Bus` switch).

When the command finishes, the tool displays a message that indicates whether the analysis passed (the ATPG process could not find any possible pattern), failed (the ATPG process found another possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis fails, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- **-Bus**

An optional switch that specifies for the tool to consider bus contention prevention during the ATPG process.

Examples

The following example creates an ATPG constraint and then checks for mutual exclusivity:

```
add_atpg_constraints 1 435
analyze_atpg_constraints 435 -exclusive
```

```
// ATPG constraint 435=1 failed mutual exclusivity check (data
// in parallel pattern 0).
```

Related Topics

[add_atpg_constraints](#)

analyze_bus

Context: dft -edt, patterns -scan

Mode: analysis

Analyzes the specified bus gates for contention problems.

Usage

```
analyze_bus {gate_id#... [-Exclusivity | -Prevention | -Zstate]} |  
-Drc_check | -All | -Auto | -ANALyze_sequentially
```

Description

Analyzes the specified bus gates for contention problems.

If the bus passes the analysis, the tool displays a message indicating that it did so. If the analysis aborts, the tool displays a message that identifies the tri-state drivers (TSDs) the tool was analyzing at abort time. If the bus fails the analysis, the tool displays a message that identifies the two offending TSDs (the tri-state drivers capable of being on simultaneously, while driving different values).

Note

 After using the analyze_bus command, the [set_gate_report Parallel_pattern 0](#) command may display conflicting values if failures are found. That is, it is possible to have TIE-X gates with a 1/0 as an output. This is due to the analyze_bus command setting values to determine contention states. This analysis continues until a conflict is found.

In the tool, analyze_bus only shows the values that were used by the ATPG, and which the command considers relevant to the bus conflict. As the purpose of the analysis is to determine TSD input combinations that result in bus conflicts, the command does not simulate the output values of the TSD.

Note

 With EDT on, Tessent TestKompress support of this command is limited: it will not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

The commands “set_contention_check On -Atpg” and “set_iddq_checks -Bus” both cause ATPG to ensure that every bus is contention free during deterministic test generation. Sometimes, this requirement cannot be met for many or all of the faults targeted by ATPG, preventing you from obtaining adequate fault coverage. When this happens, the analyze_bus command determines which bus or buses cannot be made contention free, so that you can investigate the circuit around this bus to find out what is preventing contention-free tests.

When you issue this command, you must either specify a *gate_id#* value or one of the global switches (-Drc_check or -All).

Arguments

- **gate_id#** -Exclusivity | -Prevention | -Zstate

A repeatable integer with an optional switch that specifies the identification number of the bus gate and the type of analysis you want the tool to perform.

The available switch choices are as follows:

- Exclusivity — An optional switch that specifies for the tool to analyze the bus gate to see if it has mutual exclusivity. Mutual exclusivity means that only one driver can simultaneously force a strong signal onto the bus. Exclusivity is the default behavior when you specify a *gate_id#* value without a corresponding switch.
- Prevention — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a state of non-contention.
- Zstate — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a high-impedance (Z) state.

- **-Drc_check**

A switch that specifies for the tool to run the design rule check process again to categorize all buses and display the results. This is useful if you are changing constraints or the abort limit in an attempt to pass bus checks rather than abort.

- **-ALl**

A switch that specifies for the tool to use the more extensive ATPG process to place all the fail and abort buses in a noncontentious state. The internal simulation data for this pattern is available in parallel pattern 0.

- **-AAuto**

A switch that automatically tries to locate the bus that cannot be made contention free. The analysis checks to see if any single bus cannot be made contention free. Each bus that cannot be made contention free (given the current abort limit and other restrictions) is reported to the user. Sometimes, each bus can be satisfied by itself, but some set of buses cannot all be satisfied. In this case, -Auto switch proceeds to a second analysis where it creates a minimized set of buses that can't be satisfied. The final, reduced set of buses, which cannot all be made contention free, is reported.

- **-ANalyze_sequentially**

A switch that specifies to perform a sequential analysis to find bus contention problems. For each clock defined using the [add_clocks](#) command, the sequential analysis applies a 1-cycle test, then a 2-cycle test, and so on, up to the reporting depths for which the tests fail. It then analyzes and lists the specific buses causing the failures at those depths.

Note



The same sequential analysis is performed automatically by the [analyze_restrictions](#) command if that command finds bus contention is the problem preventing success.

Examples

Example 1

The following example analyzes a bus that failed the regular bus contention checking:

```
set_system_mode analysis
analyze_bus 493
```

Example 2

The following example displays the current categorization of bus gates, and then performs the prevention check on a specific bus gate:

```
set_system_mode analysis
analyze_bus -drc_check

// ATPG bus checking results: pass=1, bidi=1, fail=0, abort=0,
// CPU time=0.00.

analyze_bus 495 -prevention

// Controllability justification was successful (data accessible using
// parallel_pattern 0).
// Pattern type: Basic_scan
```

Example 3

The following example analyzes a bus that failed the regular bus contention checking. Notice the input values of gates 7 & 8 (the two TSDs) show that ATPG has correctly illustrated a conflict condition:

```
// Warning: BUS gate /Y (9) has possible contention on drivers 7 and 8.
// (E10-1)

analyze_bus 9

// ATPG bus checking performed on /Y (9) for 2 bused TSDs.
// Failure occurred while checking TSDs 7 and 8 (data in
// parallel_pattern 0).

set_gate_report parallel 0
report_gates 9

// /Y (9) BUS
//   "I0"    I  (X)  7-/tri01/Y
//   "I1"    I  (X)  8-/tri02/Y
//   "OUT"   O  (X)  10-/Y

report_gates 7

// tri01 (7)  TSD
//   E        I  (1)  3-/E
//   "D"      I  (1)  5-
//   Y        O  (X)  9-

report_gates 8
```

```
// tri02 (8) TSD
//      E      I  (1)  4-/E1
//      "D"    I  (0)  6-
//      Y      O  (X)  9-
```

Related Topics

[analyze_restrictions](#)

[report_bus_data](#)

[set_contention_check](#)

[set_gate_level](#)

[set_gate_report](#)

analyze_compression

Prerequisites: Scan-inserted netlist

Context: dft -edt, patterns -scan

Mode: analysis

Analyzes the compression for an application in two steps.

Usage

Determine maximum possible compression (maximum chain:channel ratio)

```
analyze_compression [-Preserve_faults_from_analysis]
    [-Override_user_settings | -NO_Auto]
    [-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
    [-COVerage_effort {Low | High}]
    [-NO_REDundancy_analysis]
    [-NO_TERMINATE_ineffective_atpg]
    [-PATterns_per_pass integer]
```

Emulating a virtual single block EDT configuration

```
analyze_compression [-Preserve_faults_from_analysis]
    [-COMPACTOR_type Xpress | Basic]
    [-ENable_edt_power_controller [-MIN_Switching_threshold_percentage int]]
    [-SEParate_control_data_channels]
    [-Uncompressed_chains chain_names ...]
    [-NO_Keep_uncompressed_chains ]
    [-CHAINS N] {[ -CHANNELS M] | [{ -INPut_channels I} {-OOutput_channels O}]}
    [-Override_user_settings | -NO_Auto]
    [-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
    [-COVerage_effort {Low | High}]
    [-NO_REDundancy_analysis]
    [-NO_TERMINATE_ineffective_atpg]
    [-PATterns_per_pass integer]
```

Emulating a virtual modular EDT configuration

```
analyze_compression [-Preserve_faults_from_analysis]
    [-COMPACTOR_type Xpress | Basic]
    [-ENable_edt_power_controller [-MIN_Switching_threshold_percentage int]]
    [-SEParate_control_data_channels]
    [-Uncompressed_chains chain_names ...]
    [-NO_Keep_uncompressed_chains ]
    [-REPORT_EDT_Pins]
    {-Edt_block block1_name [-CHAINS int] {[-CHANNELS int] |
        [-INPut_channels int -OOutput_channels int]} }
    [{-Edt_block block2_name [-CHAINS int] {[-CHANNELS int] |
        [-INPut_channels int -OOutput_channels int]} } ...]
    {[ -Broadcast_all_channels_to_identical_blocks [{block1_name block2_name ...}]]} ...
```

```
{[-SHARE_data_channels [{block3_name block4_name ...}]\n    [-DATA_and_control_channels [int]]} ...\n[-Override_user_settings | -NO_Auto]\n[-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]\n[-COVariance_effort {Low | High}] \n[-NO_REDundancy_analysis]\n[-NO_TERMINATE_ineffective_atpg]\n[-PATterns_per_pass integer]
```

Description

Analyzes the compression for an application in two steps. The two steps are as follows:

1. Analyzes a scan-inserted design and returns a range of maximum chain:channel ratios where the test coverage begins to decline.
2. Emulates a hardware configuration for a specified chain:channel ratio, generates temporary test patterns, and returns test data statistics for the compression configuration.

Note

 Before using the analyze_compression command, the target design should be setup for ATPG including the definition of any controls for internal clocks and the definition of all scan chains. In most cases, you can simply replace the create_patterns command in your dofile with the analyze_compression command and the desired switches. Detailed instructions on how to use the analyze_compression command to perform compression analysis are presented in the “[Analyzing Compression](#)” in the *Tessent TestKompress User’s Manual*.

The analyze_compression command uses the fault model type specified by the [set_fault_type](#) command. If no fault model type is specified, the tool uses the “stuck” fault model by default. The analyze_compression command uses the current fault population. If no faults are added, the tool operates on all faults or a subset of sampled faults that are determined by the fault sampling rate specified by the “[set_fault_sampling_rate](#)” command.

To enable N-detection during analysis, you must first issue the [set_multiple_detection](#) command. After enabling N-detection, you can run the analyze_compression command with the -Preserve_faults_from_analysis switch and the tool preserves the N-detection information for all the detected faults; if you do not specify the -Preserve_faults_from_analysis switch, the tool restores the N-detection information for all detected faults to the states prior to the execution of the analyze_compression command. For an example of enabling N-detection, see [Example 16](#).

Note

 During emulation, if the emulated EDT block has the same name as the original one, the tool preserves all the scan cells in the original EDT block; if you specify new EDT block names, the tool tries to allocate all scan cells into the new EDT blocks to get balanced scan chains. You cannot mix the names of the original EDT blocks with the names of the newly emulated EDT blocks. This is illustrated in [Example 17](#).

For more information, see “[Analyzing Compression](#)” in the *Tessent TestKompress User’s Manual*.

Note

 When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, TestKompress assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

Arguments

- **-Preserve_faults_from_analysis**

An optional switch that preserves the fault status at the end of the analyze_compression operation based on the ATPG run of the command even though no patterns are stored. By default, without the -Preserve_faults_from_analysis switch, the fault status at the end of the analyze_compression operation is returned to its status prior to the execution of the command. This switch can be used to experiment with analyzing faults remaining after an analysis run.

If the -Preserve_faults_from_analysis switch is specified, the tool preserves the N-detection information for all the detected faults during the analyze_compression operation.

Note

 When you specify this option, if there are already internal patterns generated by a previous `create_patterns` command, the tool deletes all internal patterns and issues a warning stating that the previously generated patterns are deleted because they will be inconsistent with the fault population.

- **-Override_user_settings**

For information about this switch, see the [create_patterns](#) command.

- **-NO_AUto**
For information about this switch, see the `create_patterns` command.
- **-Compaction_effort {Low | High | Maximum | integer}**
For information about this switch, see the `create_patterns` command.
- **-NO_Analyze**
For information about this switch, see the `create_patterns` command.
- **-COVerage_effort {Low | High}**
For information about this switch, see the `create_patterns` command.
- **-NO_REDundancy_analysis**
For information about this switch, see the `create_patterns` command.
- **-NO_TERMINATE_ineffective_atpg**
For information about this switch, see the `create_patterns` command.
- **-PATterns_per_pass integer**
For information about this switch, see the `create_patterns` command.
- **-COMPACTOR_type Xpress | Basic**
An optional switch that specifies the compactor to use for the analysis. By default, the Xpress compactor is used. The specified compactor is applied to all blocks in a block-level application. This switch cannot be specified more than once; otherwise, an error message is generated.
- **-ENAble_edt_power_controller [-MIN_Switching_threshold_percentage *int*]**
An optional switch and switch-integer pair that enables emulation of the low-power controller and specifies the minimum switching threshold for the low-power controller being used for compression emulation. You must specify the `-Channel` option when using this option.

If this switch is specified, the tool uses the specified low-power controller for all blocks. If no low-power usage is specified, then no low-power is used. This switch cannot be specified more than once; otherwise, an error message is generated.

The behavior of the `analyze_compression` command, when used with this switch, can be effected by the switching threshold in several ways.
 - When using this switch, but the “`set_power_control shift`” command is not used, at the start of `analyze_compression`, the tool issues the following message:

```
// Note: Emulating low-power patterns with <N>% shift switching
// threshold.
// To use a different low power shift threshold or disable
// low power shift, use the 'set_power_control' command.
```

- When using this switch, and meanwhile the “set_power_control shift off” command is used, the tool will not emulate low-power patterns, even if specified with analyze_compression, the tool issues this message:

```
// Warning: Switching threshold for low power controller was
// specified but is ignored because 'set_power_control shift' is
// set to OFF.
// Emulating patterns without low power control.
```

- When using “-enable_edt_power_controller -min_swtinghing_threshold_percentage X”, but meanwhile “set_power_controller shift on Y is used, and X < Y, the tool uses the setting of the set_power_controller command (Y% in this case) as opposed to the hardware setting with analyze_compression (X% in this case). No message is issued.
- If set_power_control is used, but switching threshold is not used with analyze_compression, the tool issues this message:

```
// Note: Switching threshold was set using 'set_power_control
// shift on -switching_threshold_percentage <int>' but is
// ignored because the -min_switching_threshold_percentage was
// not used with the analyze_compression command to specify the
// configuration of the low power controller. Emulating patterns
// without low power control.
```

- **-SEParate_control_data_channels**

An optional switch that, if used, tells the tool to separate control and data channels for all EDT blocks during emulation, regardless of whether channel sharing is used. If this switch is not used, the tool does not separate control and data channels for all EDT blocks unless the block is in a channel sharing group specified with the -SHARE_data_channels switch.

- **-Uncompressed_chains *chain_names* ...**

An optional switch and Tcl list of chain names that specifies uncompressed scan chains. You must define these chain names with the add_scan_chains command. If a chain name is not found, the tool issues an error on the first unfound chain name.

If EDT is ON, then the following applies:

- If you specify -Uncompressed_chains without specifying the optional -NO_Keep_uncompressed_chains, then the tool treats the specified uncompressed chains and the original uncompressed chains as uncompressed chains.
- If you specify -Uncompressed_chains with the optional -NO_Keep_uncompressed_chains, then the tool only treats the specified uncompressed chains as uncompressed chains.

If the current design has no EDT or EDT is OFF, then the following applies:

- If you specify -Uncompressed_chains, then the tool only treats the specified chains as uncompressed chains while using the scan cells in the remaining uncompressed chains for compression analysis.

- **-NO_Keep_uncompressed_chains (EDT ON only)**

An optional literal that instructs the tool not to treat any original uncompressed chains in the design as uncompressed unless they are specified in the **-Uncompressed_chains**.

By default if it is an EDT design, the tool keeps any uncompressed scan chains as uncompressed and does not merge those cells with the compressed chains. If this switch is used on a design that does not contain any uncompressed chains, then the tool issues a warning.

If the current design has no EDT or EDT is OFF, then this switch is ignored.

- **-REPORT_EDT_Pins**

An optional literal that instructs the tool to call the "[report_edt_pins -all -group_by_pin_name](#)" command at the end of the compression analysis operation. The tool reports example pin names, and provides one example on how top-level pins are connected with the EDT block channels so that channel sharing/broadcasting can be emulated.

- **{-Edt_block *block1_name*} ...**

An optional, repeatable, switch and string pair that specifies the name of a design block containing compression logic (compression block). This option specifies multiple compression blocks for the analysis of a block-level architecture design.

If you only specify one compression block, the design is treated as a chip-level architecture for analysis.

Chain and/or channel specifications must be defined for each block individually.

If you do not specify any compression blocks for a block-level design, the design is analyzed with the configurations defined in the design.

Only the associated scan cells are used to analyze existing compression blocks. If you do not specify any scan channels for existing blocks, the tool uses those specified in the design.

If you specify any new compression blocks, the tool completely ignores the scan chain and channel configurations defined in the design. For the analysis, the tool distributes the scan cells across all specified blocks in a way that balances the scan chain length for all blocks. In this case, you must specify both chains and channels for each block. You cannot define both new and existing compression blocks in one design.

- **-CHAINS *int***

An optional switch and integer pair that specifies how many scan chains to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. If no scan chains are specified, the scan chains defined in the design are used.

By default, using **-chains *int*** will lead to reordering scan chains and scan cells to make the scan chains as balanced as possible. There is a way to skip the reordering.

- When you emulate a virtual single block EDT configuration:
If you do not use -chains *int*, the tool maintains the order of the scan chains and scan cells. For example the original design has 100 chains.

analyze_compression -channel 2

The tool emulates one EDT block with 2 channels and 100 chains without reordering the scan chains and scan cells.

- When you emulate a virtual modular EDT configuration:
If you do not use -chains *int* and use the original EDT block names, the tool maintains the order of the scan chains and scan cells. For example there are two blocks in the original design, blk1 (2 channels, 100 chains) and blk2 (2 channels, 200 chains).

analyze_compression -edt_block blk1 -channel 3 -edt_blk blk2 -channel 4

The tool emulates blk1 with 3 channels, 100 chains and blk2 with 4 channels, 200 chains, without reordering the scan chains and scan cells in each block.

- [-CHANNELS *int*] | [{-INPut_channels *int*} {-OUtput_channels *int*}]
An optional switch and integer pair or set of switch and integer pairs that specifies the channels to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. Channels must be specified when analyzing a design without compression logic inserted.

If no channels are specified for a design with compression logic inserted, the following scenarios apply:

- **Block-Level Architecture** — The sum of the channels defined for each block is used, and the design is emulated as a single compression block.
- **Chip-Level Architecture** — The channels defined by the existing compression logic are used.

Use the -input_channels/-output channels arguments to specify an asymmetrical channel configuration. These arguments must be used in pairs.

- {[-Broadcast_all_channels_to_identical_blocks [{block1_name block2_name ...}]]} ...
An optional argument used to construct one or more groups of identical blocks that share input channels. The specified blocks must be pre-existing identical EDT blocks. The tool will broadcast the input channels to all EDT blocks that are in one specified identical block group.

If this argument is specified but no block_name parameters are specified, the tool assumes that all existing EDT blocks are identical; in this case, only one channel broadcasting group is allowed.

- `{[-SHARE_data_channels [{block3_name block4_name ...}][
[-DATA_and_control_channels [int]]]} ...`

An optional argument used to construct one or more channel sharing groups such that the EDT blocks in each channel sharing group share data input channels with each block having individual control channels such that the total number of input channels used in this group equals the input channel number specified by -DATA_and_control_channels. The tool calculates the minimum number of input channels based on the channel sharing groups you define with this argument.

The block_name parameters are the names of the blocks defined to be in the channel sharing group.

The following rules apply to the creation of channel sharing groups:

- If no block_name parameters are specified, the design must already have two or more EDT blocks; otherwise, the tool generates an error.
- If no block_name parameters are specified, all blocks share the data channels; in this case, only one channel sharing group is allowed.
- If block_name parameters are specified, each channel sharing group must have a minimum of two blocks defined; otherwise, the tool generates an error.
- An EDT block can only be defined in one channel sharing group or one channel broadcasting group. Note, an EDT block does not have to be defined in any channel sharing or channel broadcasting group.
- If a block is defined in a channel sharing group, the [set_edt_options](#) “-separate_control_data on” switch is automatically set. In this case, the tool calculates the number of control channels that are needed for the block by determining the number of control bits and the length of data channels. The total input channel count for each block, specified by either the analyze_compression “-channels” or “-input_channels” switches, must be greater than or equal to the number of control channels plus 1. This means the block must have at least one data channel.
- The specified input pin count (N) for each channel sharing group must be greater than or equal to the sum of the #control channels on each block in the group plus 1. That is, at least one pin must be shared by the data channels of the EDT blocks in one channel sharing group.
- The number of input data pins available to be shared in the group cannot be greater than or equal to the sum of all data channels of all blocks in the group.
- The number of input data pins available to be shared in this group must be greater than or equal to the maximum number of data input channels of all EDT blocks in the group.

Note

 The analyze_compression command supports channel sharing and channel broadcasting separately. You can use it to emulate a group of cores with channel sharing and a different group of identical cores with broadcasting, but there can not be any overlap of the cores between the groups.

As a workaround, use a representative core instance from each identical core group and perform channel sharing emulation. The results should be close to adding the other equivalent cores if each core is well isolated.

Examples

Example 1

The following example analyzes a design and returns the upper range of chain:channel ratios:

```
analyze_compression
// For stuck-at faults
//
// Chain:Channel Ratio  Predicted Fault Coverage Drop
// -----
// 153                  negligible fault coverage drop
// 154                  0.01 % - 0.05 % drop
// 160                  0.10 %
// 168                  0.15 %
// 171                  0.20 %
...
// CPU time is 155 seconds.
```

Example 2

The following example emulates compression logic with 800 scan chains and 8 scan channels, generates test patterns, and displays test pattern statistics for a chip-level architecture:

```
set_fault_sampling 1
analyze_compression -chains 800 -channels 8
```

Command Dictionary (A - D)

analyze_compression

```
Statistics Report
Stuck-at Faults
-----
Fault Classes          #faults
                      (total)
-----
FU (full)           59164
-----
UO (unobserved)     167 ( 0.28%)
DS (det_simulation) 47815 (80.82%)
DI (det_implication) 10425 (17.62%)
UU (unused)          123 ( 0.21%)
TI (tied)            219 ( 0.37%)
BL (blocked)         50 ( 0.08%)
RE (redundant)       142 ( 0.24%)
AU (atpg_untestable) 223 ( 0.38%)
-----
Fault Sub-classes
-----
AU (atpg_untestable)
  SEQ (sequential_depth)   20 ( 0.03%)
  Unclassified             203 ( 0.34%)
UC+UO
  AAB (atpg_abort)        8 ( 0.01%)
  UNS (unsuccess)          159 ( 0.27%)
-----
Coverage
-----
test_coverage          99.33%
fault_coverage          98.44%
atpg_effectiveness      99.72%
-----
#test_patterns          348
#simulated_patterns      416
CPU_time (secs)          250.6
-----
```

Note: The reported statistics are based on a 1% fault sample.

```
// CPU time to analyze compression is 213 seconds.
//
// -----
// Scan volume report.
// -----
//   channels : 8
//   shift cycles : 164+4
// -----
//   pattern    # test  # scan          volume
//   type      patterns  loads  (cell loads or unloads)
// -----
//   chain_test    130    130          174720
//   basic        348    348          467712
// -----
//   total        478    478          642432  (642.4K)
//
```

Example 3

The following example emulates the specified compression logic, generates test patterns, and displays test pattern statistics for a block-level architecture:

```
set_fault_sampling 5
analyze_compression -edt_block edt_bk1 -chains 400 -channels 2 \
                     -edt_block edt_bk2 -chains 2000 -channels 10

Statistics report
-----
fault class          #faults    #faults
                  (coll.)   (total)
-----
FU (full)           21712      31930
-----
UO (unobserved)     14         26
DS (det_simulation) 17203      27194
DI (det_implementation) 4239      4374
PU (posdet_untestable) 6          7
UU (unused)          29         51
TI (tied)            22         22
RE (redundant)       112        116
AU (atpg_untestable) 87         140
-----
test_coverage        99.52%    99.47%
fault_coverage       98.77%    98.88%
atpg_effectiveness  99.94%    99.92%
-----
#test_patterns        700
#simulated_patterns  704
Note: The reported statistics are based on a 5% fault sample.

CPU time to analyze compression is 70 seconds.

// -----
// Scan volume report.
// -----
//   channels : 1
//   shift cycles : 487+49
// -----
//   pattern      # test  # scan          volume
//   type         patterns loads (cell loads or unloads)
// -----
//   chain_test    265    265             142040
//   basic         6426   6426            3444336
//   clock_po      10     10              5360
//   clock_sequential 1763   1763            944968
// -----
//   total         8464   8464            4536704 (4.5M)
```

Example 4

In the following example, a fault model type is not specified. By default, the stuck-at fault model type is used.

```
analyze_compression -chain 100 -channel 2
// ****
// * Run ATPG for stuck-at faults.
// ****
```

Example 5

In the following example, the `-Preserve_faults_from_analysis` switch is used to update the fault detection status based on the ATPG results of the `analyze_compression` command.

```
analyze_compression -chain 100 -channel 2 -preserve_faults_from_analysis
// ****
// * Run ATPG for stuck-at faults.
// ****
```

Note: Fault detection status will be updated based on the ATPG results of `analyze_compression` even though no patterns are stored.

Example 6

In the following example, the `set_fault_type` command is used to specify the fault model type to be used by the `analyze_compression` command.

```
set_fault_type transition
analyze_compression -chain 100 -channel 2
// ****
// * Run ATPG for transition faults.
// ****
```

Example 7

In the following example, the `set_fault_type` command is used to specify that a user-defined fault model type is to be used by the `analyze_compression` command.

```
set_fault_type UDFM
read_fault_sites ./fault/sampled.udfm
analyze_compression -chain 100 -channel 2
// ****
// * Run ATPG for UDFM faults.
// ****
```

Example 8

In the following example, the Xpress compactor type is specified for a single EDT block.

```
analyze_compression -compactor_type xpress -chain 100 -channel 2
```

Example 9

In the following example, both EDT blocks are emulated with the Xpress compactor.

```
analyze_compression -compactor_type xpress \
-edt_block blk1 -chain 100 -channel 2 \
-edt_block blk2 -chain 200 -channel 2
```

Example 10

In the following example, both EDT blocks are emulated with the Basic compactor.

```
analyze_compression \
    -edt_block blk1 -compactor_type basic -chain 100 -channel 2 \
    -edt_block blk2 -chain 200 -channel 2
```

Example 11

In the following example, only one EDT block is emulated with the low-power controller.

```
analyze_compression -enable_edt_power_controller \
    -min_switching_threshold_percentage 20 \
    -chain 100 -channel 2
```

Example 12

In the following example, the low-power controller is used for both EDT blocks. The switching_threshold_percentage switch is set to 20%.

```
analyze_compression -enable_edt_power_controller \
    -min_switching_threshold_percentage 20 \
    -edt_block blk1 -chain 100 -channel 2 \
    -edt_block blk2 -chain 200 -channel 2
```

Example 13

In the following example, both EDT blocks are emulated with the same low-power controller.

```
analyze_compression -edt_block blk1 \
    -enable_edt_power_controller \
    -min_switching_threshold_percentage 20 -chain 100 -channel 2 \
    -edt_block blk2 -chain 200 -channel 2
```

Example 14

In the following example, only one EDT block is emulated with the low-power controller. The switching_threshold_percentage switch is set to 15% by default.

```
analyze_compression -enable_edt_power_controller -chain 100 -channel 2
```

Example 15

In the following example, both EDT blocks are emulated with the low-power controller. The switching_threshold_percentage switch is set to 15% by default.

```
analyze_compression -enable_edt_power_controller \
    -edt_block blk1 -chain 100 -channel 2 \
    -edt_block blk2 -chain 200 -channel 2
```

Example 16

The following example enables N-detection ATPG during the analyze_compression.

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1
set_fault_sampling 2
analyze_compression -chain 100 -channel 2
```

Command Dictionary (A - D)

analyze_compression

```
// ****
// * Run ATPG for stuck-at faults.
// ****
....  
  
-----  
test_coverage                                78.79%  
fault_coverage                                 76.37%  
atpg_effectiveness                            79.44%  
-----  
#test_patterns                               617  
#simulated_patterns                          1280  
CPU_time (secs)                             61.0  
-----  
Multiple Detection Statistics  
-----  
Detections          DS Faults   Test Coverage  
(N)                  (Detection == N) (Detection >= N)  
-----  
1                      4 ( 0.29%) 1070 ( 78.79%)  
2                      9 ( 0.64%) 1066 ( 78.50%)  
3                     206 ( 14.70%) 1057 ( 77.84%)  
4                      27 ( 1.93%) 851 ( 62.67%)  
5+                     727 ( 51.89%) 824 ( 60.68%)  
-----  
bridge_coverage_estimate                      74.79%  
-----
```

Note: The reported statistics are based on a 2% fault sample.

```
// CPU time to analyze compression is 30 seconds.  
//  
// -----  
// Scan volume report.  
// -----  
// channels      : 4  
// shift cycles : 41+19  
// -----  
// pattern      # test  # scan           volume  
// type        patterns loads  (cell loads or unloads)  
// -----  
// chain_test    36     136            32640  
// basic        617    617            148080  
// -----  
// total         753    753           180720  180.7K)
```

Example 17

The following examples demonstrate how the tool enforces the naming requirements for the emulation step.

Case 1

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
-edt_block block2 -chain10 -channel 1
```

The tool maintains all cells in the original block1 and block2 because they have the same name as the original, but changes the number of scan chains and channels based on the command.

Case 2

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1 \
-edt_block block4 -chain10 -channel 1
```

The tool does not maintain all of the cells in the original blocks because the names changed. The tool reallocates cells to make all of the chains in the two blocks as balanced as possible.

Case 3

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
-edt_block block4 -chain10 -channel 1
```

The tool does not allow the mixing of the original names and the newly specified names and generates an error message.

Case 4

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1
```

The tool does not allow this case because it changes two blocks to one block, but still uses the same name as before.

Case 5

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Case 6

The original design has one EDT block named block1.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
-edt_block block2 -chain10 -channel 1
```

The tool does not allow this and displays an error message.

Case 7

The original design has one EDT block named block1.

```
analyze_compression -edt_block block2 -chain10 -channel 1 \
-edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Example 18

The following examples demonstrate how the tool supports channel sharing.

Case 1:

The design has one sharing group with 12 input channels shared between all blocks (block1, block2, block3 and block4).

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 6 \
-edt_block block4 -chain 200 -channel 2 \
-share_data_channels -data_and_control_channels 12
```

Case 2:

The design has two sharing groups. The first group has 6 input channels shared between block1 and block2. The second group has 8 input channels shared between block3 and block4.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 6 \
-edt_block block4 -chain 200 -channel 2 \
-share_data_channels block1 block2 -data_and_control_channels 6 \
-share_data_channels block3 block4 -data_and_control_channels 8
```

Case 3:

The design has one sharing group with 6 input channels shared between block1 and block2, but block3 does not share any inputs with other blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 6 \
-share_data_channels block1 block2 -data_and_control_channels 6
```

Case 4:

In this example, a block is included in multiple channel sharing groups:

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 \
-channel 6 -share_data_channels block1 block2 \
-data_and_control_channels 6 \
-share_data_channels block1 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is included in multiple channel sharing groups.
```

Case 5:

In this example, a block is added to a channel sharing group but the block does not share input channels with any other blocks in that group.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 6 \
-share_data_channels block1 -data_and_control_channels 6 \
-share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is specified inside a channel sharing group, but
// does not share input channels with any other block.
```

Case 6:

In this example, a block is specified inside more than one channel sharing group.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 6 \
-share_data_channels block1 block1 -data_and_control_channels 6 \
-share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is specified multiple times inside a channel
// sharing group.
```

Case 7:

This example assumes that each block needs two control channels.

```
analyze_compression -edt_block block1 -chain 100 -channel 3 \
-edt_block block2 -chain 200 -channel 4 \
-edt_block block3 -chain 300 -channel 4 \
-share_data_channels -data_and_control_channels 6
```

The following error is generated:

```
// Error: Channel sharing group with 'block1' (2 control channel),  
// 'block2' (2 control channel), and 'block3' (2 control channels) needs  
// at least 7 input channels for channel sharing.
```

Case 8:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. Because the command specifies to share nine channels, no channels are actually shared.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \  
    -edt_block block2 -chain 200 -channel 3 \  
    -edt_block block3 -chain 300 -channel 4 \  
    -share_data_channels -data_and_control_channels 9
```

The following error is generated:

```
// Error: Excessive input channels specified for channel sharing group  
// which includes 'block1'. This group requires between 6 and 8 input  
// channels.
```

Case 9:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. The command specifies to share five channels which is not enough.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \  
    -edt_block block2 -chain 200 -channel 3 \  
    -edt_block block3 -chain 300 -channel 4 \  
    -share_data_channels -data_and_control_channels 5
```

The following error is generated:

```
// Error: The available input channels (5) specified for the channel  
// sharing group is not enough to drive the data input channels for block  
// 'block2' and 'block3'. This group requires between 6 and 8 input  
// channels. Alternatively, if you want to keep the shared channel count  
// (5), you may reduce the number of input channels for the following  
// blocks:  
// 'block1' may have up to 2 channels.  
// 'block2' may have up to 2 channels.  
// 'block3' may have up to 3 channels.
```

Case 10:

In this example, channel sharing is specified but neither the total number of input pins nor any channel sharing groups are defined. In this case, the tool defines the channel sharing group to be all EDT blocks and calculates the required number of input channels for those three blocks.

```
analyze_compression -share_data_channels
```

The following message is generated:

```
// Sharing 8 input channels across block1 block2 block3.
```

Case 11:

In this example, channel sharing is specified and two channel sharing groups are defined. However, the total number of input pins is not specified. In this case, the tool calculates the required number of input channels for each of the two defined channel sharing groups.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
    -edt_block block2 -chain 200 -channel 4 \
    -edt_block block3 -chain 300 -channel 6 \
    -edt_block block4 -chain 200 -channel 2 \
    -share_data_channels block1 block2 \
    -share_data_channels block3 block4

// Sharing 8 input channels across block1, block2.
// Sharing 10 input channels across block3, block4.
```

Example 19

The following examples demonstrate how the tool supports broadcasting channel input to blocks defined to be identical.

Case 1:

In this example, the original design is assumed to have three identical EDT blocks and each block has eight EDT input channels. Because no block names are listed, the tools assumes all EDT blocks are identical and broadcasts to all three blocks.

```
analyze_compression -broadcast_all_channels_to_identical_blocks

// Broadcasting 8 input channels to block1, block2, block3.
```

Case 2:

In this example, the tool broadcasts to two specified identical blocks and shares channels between two specified non-identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
    -edt_block block2 -chain 100 -channel 4 \
    -edt_block block3 -chain 300 -channel 6 \
    -edt_block block4 -chain 200 -channel 2 \
    -broadcast_all_channels_to_identical_blocks block1 block2 \
    -share_data_channels block3 block4 -data_and_control_channels 10

// Broadcasting 4 input channels to block1, block2.
// Sharing 10 input channels across block3, block4.
```

Case 3:

In this example, the tool broadcasts channel inputs to two different groups of identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
                    -edt_block block2 -chain 100 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 6 \
                    -broadcast_all_channels_to_identical_blocks block1 block2 \
                    -broadcast_all_channels_to_identical_blocks block3 block4

// Broadcasting 4 input channels to block1, block2.
// Broadcasting 6 input channels to block3, block4.
```

Case 4:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of input channels.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 100 -input_channels 4 -output_channels 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2

// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of input channels.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 5:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of output channels.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 100 -input_channels 3 -output_channels 3 \
                    -broadcast_all_channels_to_identical_blocks block1 block2

// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of output channels.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 6:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of scan chains.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 120 -input_channels 3 -output_channels 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2

// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of scan chains.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 7:

In this example, Chain1 in block1 has 400 scan cells and Chain1 in block2 has 500 scan cells. The tool attempts to broadcast channel inputs to these two identical blocks. An error is generated because blocks that were specified to be identical are not identical because Chain1 in block1 has a different number of scan cells than Chain1 in block2.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
-edt_block block2 -chain 100 -input_channels 3 -output_channels 2 \
-broadcast_all_channels_to_identical_blocks block1 block2

// Error: EDT block 'block1' and 'block2' are non-identical due to
//         different number of scan cells on chain 'Chain1'.
//         '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 8:

In this example, the tool attempts to broadcast channel inputs to two different groups of identical blocks; an error is generated because one of the blocks is defined as a member of two different groups of identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
-edt_block block2 -chain 100 -channel 4 \
-edt_block block3 -chain 100 -channel 4 \
-broadcast_all_channels_to_identical_blocks block1 block2 \
-broadcast_all_channels_to_identical_blocks block2 block3

// Error: EDT block 'block2' show up in two groups."
```

Case 9:

In this example, the tool attempts to broadcast channel inputs to a block that is specified in both a group of identical blocks and a group of non-identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
-edt_block block2 -chain 100 -channel 4 \
-edt_block block3 -chain 200 -channel 5 \
-broadcast_all_channels_to_identical_blocks block1 block2 \
-share_data_channels block2 block3 -data_and_control_channels 6

// Error: EDT block 'block2' show up in two groups."
```

Example 20

In this example, there are 100 chains, named *chain1*, *chain2*, ..., *chain100* and the first four chains are set to uncompressed chains:

```
analyze_compression -uncompressed_chains chain1 chain2 chain3 chain4 -channels 4 -chain 50
```

Issuing this command specifies that *chain1* to *chain4* are uncompressed chains, while the scan cells in the remaining 96 chains are evenly distributed among 50 chains. These 50 chains are compressed with 4 EDT channels.

Example 21

In this example, there are 100 chains in an EDT design, and the first 4 chains are already uncompressed chains:

```
analyze_compression -edt_block blk1 -channels 4 -chain 50 -edt_block blk2 -channel 4 -chain 50
```

Issuing this command maintains the first 4 chains as uncompressed chains, while the scan cells in the remaining 96 chains are evenly distributed among 100 chains (50 chains in each of the two EDT block).

Example 22

In this example, there are 100 chains (*chain1* through *chain100*) in an EDT design, and the first 4 chains are uncompressed chains. To add two more uncompressed chains (*chain5* and *chain6*) during the emulation, you would issue the following command:

```
analyze_compression -uncompressed_chains chain5 chain6 -channels 4 -chain 50
```

This will treat the first 6 chains as uncompressed chains, while the scan cells in the remaining 94 chains are evenly distributed among 50 chains, and these 50 chains are compressed with EDT of 4 channels.

Example 23

In this example, there are 100 chains (*chain1* through *chain100*) in an EDT design, and the first 4 chains are uncompressed chains. To replace these uncompressed chains with new uncompressed chains (*chain5* and *chain6*) during the emulation, you would issue the following command:

```
analyze_compression -no_keep_uncompressed_chains \  
-uncompressed_chains chain5 chain6 -channels 4 -chain 50
```

This will treat *chain5* and *chain6* as uncompressed chains, while the scan cells in the remaining 98 chains are evenly distributed among 50 chains, and these 50 chains are compressed with EDT of 4 channels.

Related Topics

[set_edt_instances](#)
[set_edt_options](#)
[set_fault_sampling](#)
[set_fault_type](#)

[analyze_control_signals](#)

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Identifies and optionally defines the primary inputs of control signals.

Usage

```
analyze_control_signals [-Report_only | -Auto_fix] [-Verbose]
```

Description

Identifies and optionally defines the primary inputs of control signals.

The analyze_control_signals command identifies each control signal (clocks, set, reset, write-control, read-control, and so on) of every sequential element (DFF, latch, RAM, ROM, etc.) and optionally defines its primary input as a control signal. The purpose of analyzing the control signals is to identify the primary inputs that need to be defined as a clock, read-control, or write-control. This analysis also considers pin constraints, but only traces through simple combinational gates.

If the -Verbose option is specified, the tool issues messages that indicate why certain control signals are not reported. At the end of the analysis, the tool displays statistical information such as the number of primary inputs identified as control signals, their types, and additional information.

If the -Auto_fix option is specified, all identified primary inputs of control signals are automatically defined. For example, when a clock is identified, an implicit add_clocks command is performed to define the primary input. By default, all control signals are only reported.

This command performs the flattening process automatically, if executed prior to performing flattening.

Note

 The tool defers to your expert knowledge when you use add_clocks to define a clock; the -Auto_fix option will not alter a clock definition specified by a preceding add_clocks command.

dft -scan Details

Because it is not possible to completely identify all control signals, you should only use the results of this command as a *starting point* in creating a dofile that defines the clock control signals, read and write controls in a design. You should use this dofile when the tool is being

used for production scan and test logic insertion. Situations where the tool may not be able to identify particular control signals include the following:

- When the tool is unable to trace or simulate through complex logic, the tool cannot identify a top-level pin as a clock. These cases include clocks driven by PLLs or clock gaters that require simulating a test setup procedure to obtain a sensitizable path to a top-level pin.
- The tool cannot accurately predict the off-state of a clock by observing the flip-flops it is driving and, therefore, requires the user to verify that the off-state of the clock/set/reset lines is correct.

Because of these types of issues, this command does not identify all clocks, their off-states, or controls signals. In this case, you should explicitly add the unidentified clocks using the [add_clocks](#) command.

Arguments

- **-Report_only**
An optional literal that specifies to identify control signals only (does not define the primary inputs as control signals). This is the invocation default.
- **-Auto_fix**
An optional literal that specifies to define the primary inputs of all identified control signals as control signals. For example, when a clock is identified, an implicit [add_clocks](#) command is performed to define the primary input.
- **-Verbose**
An optional literal that specifies to display information on control signals (whether they are identified or not, and why) while the analysis is performed.

Examples

The following example analyzes the control signals, then only provides a verbose report on the control signals in the design. After examining the transcript, you can then perform another analysis of the control signals to add them.

[analyze_control_signals -verbose](#)

```
// command: analyze_control_signals -reports_only -verbose
```

```

// Begin control signals identification analysis.
-----
// Warning: Clock line of '/cc01/tim_cc1/add1/post_latch_29/WRITEB_reg/r/
// (7352)' is uncontrolled at '/IT12 (4)'.
...
...
...
// Identified 2 clock control primary inputs.
//     /IT23 (5) with off-state = 0.
//     /IT12 (4) with off-state = 0.
// Identified 0 set control primary inputs.
// Identified 1 reset control primary inputs.
//     /IRST (1) with off-state = 0.
// Identified 0 read control primary inputs.
// Identified 0 write control primary inputs.
-----
// Total number of internal lines is 105 (35 clocks, 35 sets , 35 resets,
// 0 reads, 0 writes).
// Total number of controlled internal lines is 25 (17 clocks, 0 sets ,
// 8 resets, 0 reads, 0 writes).
// Total number of uncontrolled internal lines is 80 (18 clocks, 35 sets,
// 27 resets, 0 reads, 0 writes).
// Total number of added primary input controls 0 (0 clocks, 0 sets ,
// 0 resets, 0 reads, 0 writes).
-----
```

analyze_control_signals -auto_fix**Related Topics**

[add_clocks](#)
[add_read_controls](#)
[add_write_controls](#)
[report_clocks](#)
[report_read_controls](#)
[report_write_controls](#)

analyze_drcViolation

Context: dft -edt, dft -scan, dft -testPoints, patterns -scan, patterns -ijtag, patterns -scanRetargeting, patterns -scanDiagnosis

Mode: setup, analysis

Generates a schematic in the schematic windows the portion of the design involved with the specified rule violation number.

Usage

analyze_drcViolation *rule_id-occurrence#* [-append]

Description

Generates a schematic in the schematic windows the portion of the design involved with the specified rule violation number.

When you issue the analyze_drcViolation command, the tool includes different simulation data in the netlist depending on the type of rule violation. The tool automatically opens the schematic windows of DFTVisualizer if it is not already open, and sets the gate reporting to match the type of rule violation, just as if you had manually issued the set_gate_report command. The gate report setting is shown on the status bar. For some DRC violations (E5 for example), the gate level is automatically set to primitive.

Note that the following DRC errors do not have additional information that can be displayed with DFTVisualizer:

- B1—B16
- E1, E12, E13, E14
- G rules
- K rules
- P rules
- T1, T8—T11, T13—T15, T18—T23
- W1—W19, W32, W33

For more information about displaying DRC information, refer to the [report_drcRules](#) command. For information about each DRC, refer to “[Design Rule Checking](#)” on page 2639.

Arguments

- ***rule_id-occurrence#***

A literal and integer argument pair that specifies the exact design rule violation (including the occurrence) that you want to analyze. The tool traces the violation back to the probable cause and then the schematic viewer displays all the gates in that trace. The argument must

include the design rules violation ID (*rule_id*), the specific occurrence number of that violation, and the hyphen between them. For example, you can analyze the second occurrence of the C3 rule by specifying C3-2. The tool assigns the occurrences of the rules violations as it encounters them, and you cannot change either the rule identification number or the ordering of the specific violations.

- **-append**

An optional switch that specifies to append the new schematic to the existing contents of the schematic windows instead of replacing the contents. If you do not specify the **-append** switch, the existing schematic is deleted and new data is shown in the schematic windows.

Examples

The following example defines the off-state of a clock incorrectly, causing a C2 rule violation. When a rule violation occurs, you can use the schematic windows to analyze the probable cause of the error.

With this example, the schematic windows display the sequential element associated with the clk input, along with a backward trace through the gates and nets to the associated primary input.

```
add_clocks 0 clk
set_system_mode analysis

...
// -----
// Begin scan clock rules checking.
// -----
// 1 scan clock/set/reset lines have been identified.
// All scan clocks successfully passed off-state check.
// Error: Clock /CLK cannot capture data with other clocks off. (C2-1)

analyze_drcViolation c2-1
```

Related Topics

[open_visualizer](#)
[report_drc_rules](#)
[set_drc_handling](#)
[set_visualizer_preferences](#)

analyze_fault

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in DFTVisualizer.

Usage

Struck/Transition Usage

```
analyze_fault pin.pathname {-Stuck_at {0 | 1} | -Rise | -Fall}  
[-Observe gate_id#] [-Boundary] [-AUto] [-Continue] [-Display [APPend]]
```

Path Delay Usage

```
analyze_fault path_name {-Rise | -Fall} [-Observe gate_id#] [-AUto] [-Display [APPend]]
```

Description

Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in DFTVisualizer.

The analyze_fault command performs an analysis to identify why the fault that you specify was not detected. You can use the -Observe switch to specify the observe point for the sensitization analysis.

If you are using DFTVisualizer, you can specify the -Display switch to graphically display the relevant circuitry for any fault detection. This may assist you in identifying either why a fault wasn't detected or how a fault was detected.

The fault analysis performed by the analyze_fault command consists of the following actions:

1. A message is given if the selected fault has been nofaulted.
2. A message is given that will identify if the fault is in the current fault list. If the fault is in the current fault list and the fault is the representative member, its fault classification shows the fault class and the subclass information.
3. If the fault was not identified as included in the active fault list, the basic fault analysis is performed that determines if the fault can be classified as unused, tied, blocked, or detected by implication.
4. If the fault category is detected by simulation, detected by implication, or unused, a message is given and the analysis is terminated. You can override the termination by using the -Continue switch.
5. If the fault category is tied, all sources of the tied condition are identified and the analysis is terminated.

6. If the fault category is blocked, all blockage points (100 maximum) are identified. For each blockage point, all sources of the tied conditions causing the blockage are identified. The analysis then terminates.
7. The states that result from all constrained pins and stable non-scan cells are calculated.
8. If the fault site is now prevented from attaining the necessary state, a message is given indicating the fault is tied by constrained logic, all sources of the tied condition are identified, and the analysis then terminates.
9. An analysis is made to identify all blockage points (100 maximum) and all potential detection points (25 maximum).
10. If there are no potential detection points, the blockage points are identified and for those points blocked by tied logic, all sources of the tied condition are identified. The analysis then terminates.
11. If there were potential detection points, the detection points are identified (25 maximum).
12. A controllability test generation is performed to determine if the fault site can be controlled. If successful, the test generation values are displayed using parallel_pattern 0. If unsuccessful, the analysis then terminates.
13. If an observe point is not selected, a complete test generation is attempted where the fault is sensitized from the fault site to any unblocked point. Potential problem points in any sensitization path are identified. The points will include tri-state driver enable lines, transparent latch data lines, clock_pos, wire gates, latch and flip-flop set/reset/clock lines, RAM write/read/address/data lines, and ROM read/address lines. If the test generation is successful, the test generation values are displayed using parallel_pattern 1.
14. If an observe point is selected, the fault is sensitized from the fault site to the observe point. Potential problem points in the sensitization path are identified. If the sensitization is successful, the test generation values are displayed using parallel_pattern 1.

Note

 Only stuck-at, path delay, and transition fault types can be analyzed using the analyze_fault command.

When the fault type is path delay, the analyze_fault command performs an unsensitizable path analysis when the ATPG run is unable to create a test pattern. If the analysis finds that some segment of the path is unsensitizable, it attempts to find a minimum number of gates in the path which are required to prove the path unsensitizable.

For example:

analyze_fault path37 -s 0

produces the following report:

```
// -----
// Path delay fault analysis for path37 slow to rise on
// launch point = 22032, capture point = 167521
// -----
// Path delay test generation not successful for capture point = 167521
// due to atpg_untestable.
// Begin unsensitizable path analysis for path=path37
// Cycle=0 path sensitization check failed status=redundant
// Simultaneous sensitization of 2 path segments cannot be achieved
// (status=redundant).
// These are:
//      OR   145279 (I1) + ;
//      ...
//      AND  150252 (I1) + ;
// Path is a static unsensitizable path. Robust detection will be
// impossible.
```

The analyze_fault report shows that it is impossible to use both input 1 (the second input, inputs are numbered starting at 0) of the OR gate 145279 while at the same time using input 1 of the AND gate 150252. The ellipsis (...) indicates that there are other gates in the path between the reported gates which are not relevant to the unsensitizable path problem.

The “[delete_fault_sites](#) -Unsensitizable_paths” command lets you delete proven unsensitizable paths.

When the fault type is transition, the analyze_fault command performs the following analysis steps:

1. The controllability of logic value 0 and 1 at the fault site is analyzed.

If the analysis succeeds, the fault site is controllable. You can access the analysis results by setting the gate report type to be parallel pattern 0(2) to set logic value 1(0) at the fault site for a slow-to-rise fault, or logic value 0(1) at the fault site for slow-to-fall fault.

If the tool aborts this step or it otherwise fails, the fault cannot be detected due to a lack of controllability at the fault site. The tool then terminates the analysis unless you force it to continue.

2. The ability to launch a slow-to-rise (0 -> 1) or slow-to-fall (1 -> 0) transition at the fault site is analyzed.

If the analysis succeeds, the desired transition can be created at the fault site. You can access the analysis results by setting the gate report type to be parallel pattern 3.

If the tool aborts this step or it otherwise fails, the fault cannot be detected because a transition cannot be launched at the fault site. The tool terminates the analysis unless you force it to continue.

3. The tool attempts to generate a test pattern for the transition fault being analyzed.

If successful, a test pattern exists to detect the fault. You can access the analysis results by setting the gate report type to be parallel pattern 1 and the analysis process terminates.

If the tool aborts this step or it otherwise fails, the tool will proceed to the next step.

4. The tool attempts to generate a test pattern for a stuck-at fault at the same fault site.

If successful, there exists a sensitive path from the fault site to an observation point. The reason for aborting or failing to generate the test pattern for the original transition fault is that launching a transition at the fault site may block the fault propagation path. You can access the analysis results for the stuck-at fault by setting the gate report type to be parallel pattern 4.

Arguments

- ***pin.pathname***

A required string that specifies the pin pathname of the fault where you want to perform stuck/transition analysis.

- ***path.name***

A required string that specifies the pathname where you want to perform path delay analysis.

- **-Stuck_at 0 | 1**

A switch and literal pair that specifies the stuck-at fault value that you want to analyze. The stuck-at values are as follows:

0 — A literal that specifies that the tool analyze the *pin.pathname* for a “stuck-at-0” fault.

1 — A literal that specifies that the tool analyze the *pin.pathname* for a “stuck-at-1” fault.

- **-Rise**

A switch that specifies a slow-to-rise transition fault.

- **-Fall**

A switch that specifies a slow-to-fall transition fault.

- **-Observe *gate_id#***

An optional switch and integer pair that specifies the observe point for the sensitization analysis.

gate_id# — An integer that specifies a gate identification number whose location you want to use as the observe point for the sensitization analysis.

- **-Boundary**

An optional switch that specifies to display the boundary faults when analyzing an ATPG untestable, tied, blocked, or redundant fault.

- **-AUto**

An optional switch, applicable only to path delay faults, that automatically determines how far along the path the tool can successfully propagate a path delay fault.

- **-Continue**

An optional switch that forces the tool to complete the analysis of faults that have already been detected by the pattern set. This lets you inspect the generated pattern by using the report_faults command.

If you do not specify this switch and the tool detects the fault category by simulation, by implication, or as unused, then the tool displays a message and terminates the analysis.

- **-Display [APPend]**

An optional switch and literal pair that specifies for the tool to create a gate list relevant to the fault being analyzed and, if DFTVisualizer is open, to automatically update the schematic in the Flat Schematic window with the information; if the Append switch is specified the schematic is appended to the existing contents of the Flat Schematic window instead of replacing the contents.

The schematic view contains annotations for the following cases as required:

- Successful ATPG — All gates that sensitize the fault effects to an observable point are added to the gate list. The pins on these gates are annotated with the simulated value that results from the pattern that was created.
- Uncontrollable Fault Site due to Forbidden Conditions — All gates that prevent the fault site from attaining the required state are added to the gate list. The pins on these gates are annotated with the constrain_value data.
- Blocked Fault Site due to Forbidden Conditions — All gates that sensitize the fault effects to the blockage point and all gates that prevent the blockage points from attaining the necessary values are added to the gate list. The pins on these gates are annotated with the constrain_value data.

Examples

The following example performs test pattern generation, then performs an analysis to determine why the tool did not detect a stuck-at-1 fault at pin /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q:

```
set_system_mode analysis
create_patterns
report test data -class au

// fault analysis summary of 17 faults
// number faults tied by constraints = 7
// number faults connected to nonscan_latch = 15
// number faults connected to clock/set/reset = 8
// number faults connected from clock = 6
// number faults connected from nonscan_latch = 7
// number faults unclassified = 0
```

report_faults -class au

type	code	pin.pathname
0	AU.TC	/tcb_tck
1	AU.TC	/tcb_tck
0	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/SD
1	AU.PC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/SD
0	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TE
1	AU.PC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TE
0	AU.PC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/D
1	AU.PC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/D
0	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TI
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TI
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q
0	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/CP
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/CP
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/Q
0	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/CP
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/CP
1	AU.TC	/ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/D

analyze_fault /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q -stuck_at 1

```

// -----
// Fault analysis for /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2
// (29217) output Q stuck at 1
// -----
// Current fault classification = AU.TC
// Fault site sequential depth: Control_0 = 255, Control_1 = 0, Observe = 255.
// Fault site is tied to 1 due to constrained logic.
// Tied gate source: 1 /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2
// (29217)

```

Related Topics

[delete_fault_sites](#)
[report_faults](#)
[report_testbench_simulation_options](#)

analyze_graybox

Context: patterns -scan

Mode: analysis

Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to “true”. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.

Usage

```
analyze_graybox [-no_reset] [-preserve_instances instance_objects]
                [-max_sequential_levels {number | unlimited}] [-collect_reporting_data]
```

Description

Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to “true”. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.

Note

 All module names, except the top-level module, are uniquified by adding a prefix in the generated graybox netlist. You can change the default prefix by using the -edited_module_prefix option of the set_insertion_options command.

Modules with the exclude_from_concatenated_netlist attribute set to true are not included inside the graybox netlist and their module names are not changed. This allows you to still bind the real simulation model of the hard macros such as memories or PLL models during simulation with the gray box views. However, even if the exclude_from_concatenated_netlist attribute is set to true, child physical region modules will be uniquified and pruned into the parent block’s graybox. A child physical region module is recognized with the is_physical_module attribute and/or is_graybox_module attributes being set to true. In the rare case that the child physical block is not loaded using the read_design command AND there is no ICL module associated with it, you will have to set the is_physical_module attribute manually if you want it to be uniquified and pruned into the parent block’s graybox.

Before issuing the analyze_graybox command, you must place the design in external mode. That is, the output wrapper chains should be set up in non-capture mode, whereas the input wrapper chains are set up in both capture and shift (or hold or rotate) modes. At the end of the analysis, a summary of the identified instances displays in the session transcript.

The graybox analysis performs the identification by tracing backward from all primary output pins, wrapper chain sequential cell input pins and specified preserve instance input pins. In general, the backward tracing used by the graybox analysis considers a path sensitization that is based on the state-stability values obtained by simulating the pin constraints and test-setup procedures provided for the external test mode. However, a pure structural tracing method is used when tracing backward from preserve instance input pins.

You should exclude the scan-out pins of the core chains from the backward tracing to prevent marking internal (core) logic for graybox. You accomplish this by setting the ignore_for_graybox attribute of the scan-out pins to “true” using the [set_attribute_value](#) command.

The -preserve_instances option can be used to include arbitrary instances (the instantiations of modules or the instantiations of library models) in the graybox logic. The tool bypasses the graybox analysis within specified preserve module instances and automatically includes all of their sub-instances and nets in graybox. This allows you to overcome the limitations that the sensitized path based backward tracing imposes, such as not allowing complex test-setup procedures that initialize non-scan cells in clocks paths, limiting the sequential depth, etc.

One typical scenario to use the -preserve_instances option is when the design includes a user-defined clock control module. The entire module can be specified as a preserve instance if it cannot be identified by the graybox analysis.

Certain categories of instances are automatically added to the list of preserved instances during graybox generation.

- Instances of EDT logic identified by EDT Finder as well as EDT logic in the channel pipeline stages are automatically preserved, as EDT Finder is enabled by default. If you have disabled EDT Finder, use the -preserve_instances option to preserve these instances.
- All instrument instances that you define using the [add_core_instances](#) command are automatically preserved. For more information, see “[Tessent Core Description \(TCD\)](#)” in the *Tessent TestKompress User’s Manual*.

Note

 This only applies if EDT is used in the external mode so EDT is defined in the current mode when performing graybox analysis. If an EDT controller in the core is not used for external test, it does not need to be preserved in the external mode.

- If you are using IJTAG and have an ICL description of the current design, then any ICL instance that is identified as part of the scan test network and which can be mapped to a design instance is automatically preserved. An ICL module is identified as a Scan Resource Instrument when it has the ICL module attribute called “[keep_active_during_scan_test](#)” set to “true”. If you create the scan instrument using the Tessent tools (for example, an EDT controller), and insert the IJTAG network using the [create_dft_specification](#) and [process_dft_specification](#) commands, then the keep_active_during_scan_test attribute is already added by the tool. Otherwise, you must manually add the keep_active_during_scan_test attribute. For more information, see “[IJTAG Network Insertion](#)” in the *Tessent IJTAG User’s Manual*.

Arbitrary instances and arbitrary nets in the design can be marked for graybox netlist generation by manually setting their in_graybox attributes to “true” using the [set_attribute_value](#)

command. These attributes are reset to “false” by the `analyze_graybox` command unless you specify the `-no_reset` switch.

Note

 Manually marking instances and nets for graybox netlist is not recommended because the tool does not automatically mark the nets and instances between such instances and the instances identified through the graybox analysis. Besides, the manual marking of instances and nets should be performed hierarchically to cover the parent instances and nets in order to properly synthesize them in the generated graybox netlist.

Currently, only Mux-DFF scan architecture is supported with the graybox functionality.

For more information about graybox concepts and procedures, refer to “[Graybox Overview](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- `-no_reset`
An optional switch that specifies not to reset the previously set `in_graybox` attributes of all design instances, and set additional ones identified by the graybox analysis. By default, the tool resets all `in_graybox` attributes when the command is issued without this switch.
- `-preserve_instances instance_objects`
An optional switch with a Tcl object list that automatically includes instances in the graybox netlist without having to deploy the graybox analysis tracing routines on these instances and the sub-instances contained within. The connections to and from the preserved instances are still determined by the analysis to include in the graybox netlist. You can also declare the specified instance objects as blackbox module instances before the analysis.
- `-max_sequential_levels {number | unlimited}`
An optional switch and integer or literal pair that specifies the allowed number of levels of non-scan cells when tracing back from the POs or from the data input pins of the wrapper cells before reaching the PIs or other wrapper cells. If the specified sequential levels limit is exceeded during backtracing, all the logic so far traced from the origin pin is dropped and not included in the graybox netlist. The limit specified by this switch does not apply to the backtracing from the clock, set, reset and scan enable pins of the wrapper cells. Sequential cells initialized to constant values (Tie-0/1) with an active set/reset pin are not counted as sequential levels during backtracing. The “*number*” argument can accept a value of 0 or greater, where 1 is the default value when this switch is not specified with the command. The “unlimited” argument specifies to not limit the number of sequential levels during backtracing.
- `-collect_reporting_data`
An optional switch that specifies to count the number of leaf instances (library model instances at design level) that are marked for each backward tracing origin, such as a PO, a wrapper cell input pin, or a “*preserve instance*” input pin (all input pins combined). This

switch is required if you want to use the [report_graybox_statistics](#) command to report the collected data. Note that collecting reporting data requires additional netlist tracing and could impact analysis execution performance.

Examples

The following command performs graybox analysis on the current design. The module instance “tap_ctrl_ins” and all sub-instances within are marked for graybox logic without going through the graybox analysis.

```
analyze_graybox -preserve_instances tap_ctrl_ins -collect_reporting_data
report_graybox_statistics -top 10
```

Related Topics

[report_graybox_statistics](#)
[set_attribute_value](#)
[write_design](#)

analyze_restrictions

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis to automatically determine sources of the problem from a failed ATPG run.

Usage

All Fault Types Except IDDQ Faults Usage

```
analyze_restrictions [-Auto] [-COnflict_source_analysis]
```

IDDQ Faults Usage

```
analyze_restrictions [-INT_float_auto_except]
```

Description

Performs an analysis to automatically determine sources of the problem from a failed ATPG run.

The analyze_restrictions command reports the ATPG restrictions that caused a failed ATPG run. If you issue the analyze_restrictions command without an argument, the default is -Auto. The analysis may be lengthy depending on the number of restrictions. You can stop the analysis at any time by pressing Ctrl-C.

Arguments

- **-Auto**

An optional switch that automatically performs the following step-by-step analysis:

- **Step 1** — The tool identifies the type of constraints that failed the analysis.
- **Step 2** — The tool analyzes the problem constraint types and finds the minimum constraints that prevent the tool from passing the analysis.
- **Step 3** — Optional. The tool performs a detailed analysis of each constraint identified in Step 2. You perform this detailed analysis by using the -COnflict_source_analysis switch in conjunction with the -Auto switch.

The set of constraints that conflict with the constraint are reported. The tool analyzes the constraints in the following order:

- Dynamic ATPG constraints
- Dynamic cell constraints
- Static ATPG constraints
- Static cell constraints
- Named capture procedure

- Bus floating checking
 - Bus contention (with the specified driver restriction)
 - Port contention
- -Conflict_source_analysis
 - An optional switch that performs a detailed analysis (Step 3) of each constraint identified in Step 2 in the -Auto switch description. This option can increase the tool's runtime under certain circumstances.
 - -INT_float_auto_except
 - An optional switch that creates Iddq exceptions automatically for the “-int_float” Iddq check failure sites found during the analysis. Iddq exceptions are defined for failures that are the result of “-int_float” Iddq checks previously specified with the set_iddq_checks command. You can report the automatically added exceptions with the report_iddq_exceptions command or use the delete_iddq_exceptions command to remove unwanted exceptions.

Examples

Example 1

The following example analyzes a failed ATPG run in which a user-specified ATPG constraint is conflicting with a cell constraint and causes the ATPG to fail test generation.

create_patterns

```
// Error: Current restrictions cannot be simultaneously satisfied with
// ANY abort limit. Status=atpg_untestable.
// Halting run. Suggest analyze_restrictions.
// You can specify the -NOAnalyze option in command "create_patterns"
// to force the run, but coverage will probably be low.
```

analyze_restrictions

```
// -----
// | Step-1 Global constraint analysis |
// -----
// Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
// -----
// | Step-2 Identifying minimum conflicting constraint |
// -----
// Following atpg constraints conflict with the rest of constraints:
//   atpg constraint: 1 dynamic /FF3/Q (10)
// You can use -conflict_source_analysis switch to analyze which
// constraints conflict with the above constraints.
```

To perform detailed analysis of the conflict, then issue the analyze_restrictions command with the optional -conflict_source_analysis switch:

analyze_restrictions -conflict_source_analysis

```
// | Step-1 Global constraint analysis |
// -----
// Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
// -----
// | Step-2 Identifying minimum conflicting constraint |
// -----
// Following atpg constraints conflict with the rest of constraints:
//   atpg constraint: 1 dynamic /FF3/Q (10)
// -----
// | Step-3 Identifying conflict source of first problem constraint |
// -----
// Following constraint conflicts with problem constraint #1:
//   atpg constraint: 1 dynamic /FF1/Q (8)
```

Example 2

The following IDDQ example analyzes a failed ATPG run, displays the results of the analysis, and automatically adds an Iddq exception for each ZVAL gate that failed in an “-int_float” Iddq check. Only part of the report, about Iddq failures, is shown.

```
analyze_restrictions -int_float_auto_except

...
// With Iddq constraints and Iddq checks removed, PASSES ability to
// satisfy restrictions.
// Problem is with Iddq specific restrictions. Analyzing further to find
// problems.
// With Iddq constraints removed, and all Set Iddq Check options removed
// except -intfloat, FAILS.
// Must fix -intfloat option problems to generate ANY iddq test given
// current restrictions & circuit.
// Will attempt to locate and report specific -int_float violators.
// No single bus is constrained to Z (or already Z). Will analyze
// further.
// The following ZVAL gates cause -int_float to fail by themselves.
//   ZVAL gate /cpu/alu/U2392/ (54417). ABORTed.
//   ZVAL gate /cpu/alu/U4177/ (63712). ABORTed.
// Will add_iddq_exceptions for -int_float failure sites.
...
```

Related Topics

[add_iddq_exceptions](#)
[analyze_atpg_constraints](#)
[delete_iddq_exceptions](#)
[report_iddq_exceptions](#)
[set_contention_check](#)
[set_iddq_checks](#)

analyze_scan_chains

Context: dft -scan

Mode: analysis

Distributes scan elements into new scan chains.

Usage

```
analyze_scan_chains [-force]
```

Description

This command distributes scan elements into new scan chains; scan elements identified as non-scannable are not considered for distribution. The scan modes and chain families defined using the [add_scan_mode](#) and [create_scan_chain_family](#) commands respectively control how the allocation of new chains is done. If not explicitly specified, the tool automatically defines a default unwrapped scan mode.

When allocating new scan chains, this command considers various scan element attributes such as “is_non_scannable”, “clock_domain”, “clock_edge”, “class”, “cluster_name”, and “power_domain_name” along with implicit/explicit distribution constraints imposed by the [set_scan_insertion_options](#), [add_scan_mode](#), and/or [create_scan_chain_family](#) commands.

The resulting scan chain objects are themselves scan elements that can be introspected; they do not have their “exists” attribute set as they are not implemented in the netlist yet. These scan chains contain the distributed scan elements, but not the final required pipelining, retiming, or mode-enabling logic — that gets added when the [insert_test_logic](#) is invoked, once the distribution model is acceptable to the user.

Arguments

- **-force**

When no scan modes have been defined, this optional switch deletes the results of previous wrapper cell analysis and infers a default unwrapped scan mode.

Examples

analyze_scan_chains

```
// Chain allocation of 'int_test' mode completed:  
//      14 distributed chains of sizes ranging from 12 to 13  
// Chain allocation of 'ext_test' mode completed:  
//      5 distributed chains of sizes ranging from 10 to 13  
...
```

report_scan_chains

analyze_simulation_mismatches

Context: patterns -scan

Mode: analysis

Analyzes simulation mismatches that occur between test patterns and a simulation performed by a specified external timing simulator.

Usage

```
analyze_simulation_mismatches {AUTO
| {-Failure_file failure_file [-VCD_file vcd_filename]}
| {-MISmatches mismatch_bit...} | Verilog_testbench_pathname
| [-Internal_patterns | -External_patterns]
{[-SImulation_script simulator_invoke_script] | [-MAnual]}
[-SCan_test | -CHain_test | -All_test] [-HOST server_name] [-Lib lib_name]
[-TEstbench_for_vcd new_filename [-REplace]] [-SHift_check {ON | OFF}] [-VERbose]
```

Description

Analyzes simulation mismatches that occur between test patterns and a simulation performed by a specified external timing simulator.

By default, the simulation VCD file, test patterns, and design data are all analyzed and the mismatch sources are identified. Once the analysis is complete, you can use DFTVisualizer to graphically display the overlapping data and pinpoint the source of each mismatch.

Refer to “[Potential Causes of Simulation Mismatches](#)” in the *Tessent Scan and ATPG User’s Manual* for complete information.

Note

 All generated files are placed in the *work_dft_debug* directory inside your working directory. This directory is created if it does not already exist.

When using the `analyze_simulation_mismatches` command for a pattern with a P53 violation, the tool issues an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```

The problem of the P53 error is due to an undefined clock waveform in the used timeplate. Refer to “[P53](#)” on page 2940 for additional information on resolving this issue.

Note

 The timestamp where the simulation actually fails does not match what this command reports, because the time `analyze_simulation_mismatches` command reports includes test setup to the beginning of the failing cycle.

Arguments

- **-Auto**

Required switch that automatically runs the entire simulation mismatch analysis on the currently loaded test patterns and design. This is the default behavior.

The top-level design and Verilog library must be compiled in your work directory prior to starting the simulation mismatch analysis. The tool will search for a “work” directory within the current working directory for these compiled files. Use the -Lib switch to specify a different directory.

The tool uses ModelSim as the Verilog simulator unless you specify a different simulator with the -Simulation_script switch.

- **-Failure _file *failure_file***

Required switch and string pair that specifies the pathname to an existing failure file to use for the simulation mismatch analysis. Using this switch bypasses Stages 1 and 2 of the analysis.

-VCD_file *vcd_filename*

Optional switch and string that specifies the pathname of an existing simulation results VCD file for the mismatch analysis. This switch can only be used with the -Failure_file switch. Using this switch bypasses the debug test bench simulation performed in Stage 4.

- **-MISmatches *mismatch_bit...***

Required switch and repeatable string that specifies a list of mismatches to analyze instead of using a failure file.

A *mismatch_bit* should be specified in one of the following formats:

fail_pattern_index chain_name cell_index

For example: “1 chain5 20” specifies the mismatch is on scan pattern 1, scan chain5, scan cell index 20.

fail_pattern_index PO_name

For example: “2 xyz” specifies the mismatch is on pattern 2 at PO signal xyz.

- **Verilog_testbench_pathname**

A required string that specifies the pathname of an existing Verilog test bench for the invoking the old mismatch analysis. This option supports only ModelSim and performs an interactive communication to the Verilog simulator instead of dumping VCD data

- **-Internal_patterns**

An optional switch that specifies the current internal pattern set for the mismatch analysis. This is the default.

- **-External_patterns**
An optional switch that specifies the currently loaded external pattern set for the mismatch analysis.
- **-SImulation_script *simulator_invoke_script***
An optional switch and string pair that specifies a simulation invocation script for the mismatch analysis. The script must invoke an external simulator and compile and simulate the test bench specified by the -PAtterns switch. By default, the analysis uses the simulator specified by the [set_external_simulator](#) command.
- **-MAnual**
An optional switch to specify the “analyze simulation mismatch” functionality to stop at the step when the external Verilog simulator is needed. This option is useful when the Verilog simulation environment is complex and the simulation script cannot be written directly. When specifying the step, the tool stops after saving Verilog test bench and gives an instruction for what to do next to complete the mismatch debugging flow. See Example 6.

Note

 Note that the -MAnual and -SImulation_script switches are mutual exclusive.

- **-SCan_test**
An optional switch that specifies that the tool only analyzes scan patterns. This is the default.
- **-CHain_test**
An optional switch that specifies that the tool only analyzes chain patterns.
- **-ALl_test**
An optional switch that specifies that the tool analyzes both scan and chain patterns.
- **-Lib *lib_name***
An optional switch and string pair that specifies the pathname of the working simulation library. The working simulation library contains the compiled top-level design and any required parts libraries. By default, the tool uses the library named *work* in the same directory as the pattern test bench. For ModelSim applications, you must use this switch to specify a different library whether or not you enter the *vlib* command.
- **-HOST *server_name***
An optional switch and string that specifies the name of a remote host on which to run the simulation portions of the mismatch analysis. The specified server must be accessible via rsh. For more information on rsh requirements, see “[Multiprocessing Requirements](#)” in the *Tessent Scan and ATPG User’s Manual*.
- **-TEstbench_for_vcd *new_filename***
An optional switch and string that specifies a file name for the debug test bench created in Stage 3 (default name *mentor_default.v_vcdtb.v*). The name of the simulation results VCD

file is derived from the name specified by this switch. The default VCD dump file name created from the debug test bench is *mentor_default.v_debug.vcd*

- **-REplace**
An optional switch that overwrites the *debug_testbench_filename* file if it already exists.
- **-SHift_check {ON| OFF}**
An optional switch and literal that allows you to enable/disable checking for simulation mismatches that occur during the scan shift cycles. By default, the tool performs shift checking, but you might want to disable this feature for performance reasons after you are sure that shift operation is working properly.
 - ON** — Checks and reports if the scan shift cycle contains a mismatch between the Verilog simulation value and the ATPG expected value.
 - OFF** — Analyzes mismatches only on the capture cycles.
- **-VERbose**
An optional switch that displays additional information about the simulation commands run in the background and the detailed mismatch data.

Examples

Example 1

The following example assumes the pathname of the ModelSim bin directory is defined by the PATH variable and that patterns are generated but not yet saved. This example demonstrates the automatic simulation mismatch analysis flow—see “[Automatically Analyzing Simulation Mismatches](#)” in the *Tessent Scan and ATPG User’s Manual*.

The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates a Verilog library named *my_mismatch_work*, compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. DFTVisualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

Figure 3-23. analyze_simulation_mismatches Example

```
set_external_simulator vsim -c
write_patterns results/pat.p.v -verilog -parallel -replace
sys vlib my_mismatch_work
sys vlog -work my_mismatch_work my_design_source/my_design.v -v my_verilog_source/verilog.lib.v

Model Technology ModelSim SE vlog 5.5a Compiler 2001.04
-- Compiling module DFFS_scan
-- Compiling module my_top_level_module
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling module xor3
-- Compiling module mux21
...
-- Compiling module buff
-- Compiling module mgc_latch
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling UDP ud_dff
-- Compiling UDP ud_dlat

Top level modules:
my_top_level_module

analyze_simulation_mismatch -auto -internal -lib my_mismatch_work
//New directory work_dft_debug has been created to store files for mismatch debug.
// Previous simulation mismatch sources are deleted.
//-----
//Step-1 Creating Verilog testbench work_dft_debug/mentor_default.v with failure file
//dump.
//-----
//Step-2 Running Verilog testbench work_dft_debug/mentor_default.v to create failure file.
//-----
//Step-3 Loading failure file work_dft_debug/mentor_default.v.fail.
//-----
//Step-4 Creating Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v with VCD dump.
//-----
//Step-5 Running Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v to create VCD
//file.
//Running ModelSim to create VCD file work_dft_debug/mentor_default.v_debug.vcd
//-----
//Step-6 Loading VCD file work_dft_debug/mentor_default.v_debug.vcd.
//-----
//Step-7 Analyzing 25 internal scan patterns in parallel format.
//-----
//ID instance (gate_id) pattern-ix time(ns) seq_depth ATPG_val VCD_val mismatch_reason
//----- -----
// 1 /ix1286/Y (364)    scan-1      1900      0       1       0 incorrect library cell "and4"
// 2 /ix1278/Y (265)    scan-15     17300      0       1       0 incorrect library cell "and4"

report_mismatch_sources -display debug data wave
```

Example 2

This example demonstrates using the `-failure_file` switch when you have previously done Verilog simulation. This example assumes you have run an initial simulation to verify test patterns, generated a failure file, and created a script to set up and run an external simulator on a separate server from the ATPG session. The following example runs a simulation mismatch analysis on the specified failure file using internal test patterns on a separate server.

analyze simulation mismatch -failure_file pat.fs.v.fail -simulation_script runsim

```
// Running script "runsim pat.fs.v.fail_vcdtb.v" from local host to create VCD file pat.fs.v.fail_debug.vcd.
// Debug 24 internal patterns, CPU time = 0.02 sec
// -----
// Step-5 Analyzing 24 internal scan patterns in parallel format.
// -----
// ID instance (gate_id) pattern-ix time (ns) seq_depth ATPG_val VCD_val mismatch_reason
// ----- -----
// 1 /ix1286/Y (364)    scan-1      1900      0       1       0 incorrect library cell "and4"
// 2 /ix1278/Y (265)    scan-15     17300      0       1       0 incorrect library cell "and4"
```

For information on the output of this command, see the `report_mismatch_sources` command.

Example 3

This example shows the results of using the `-failure_file` and `-vcf_file` switches and shows a simulation script “runsim” for ModelSim:

```
#!/bin/csh -f
#Expect netlist has been compiled to work directory, next line commented
#vlog ~design/netlist.v -v ~/library/lib.v
vlog ${1}
vsim ${2} -c -do "run -all" -sdfmax /${2}/design_inst=~/design/good.sdf
```

Note that in the previous example, `${1}` is the Verilog test bench name. The simulation script is used twice, first for running the test bench with the initial pattern set to get the failing bits, and then a second run for the reduced test bench to get the VCD file. Therefore, the vlog script needs to take a parameter instead of fixed test bench name. The `${2}` is the Verilog test bench top instance name, which may change when the test bench name changes, so the name needs to be a parameter rather than a fixed name.

The following example uses the “rnsim” script for automatic simulation mismatch analysis. Assume that the design and Verilog library have been compiled in advance in the work directory. After creating patterns, execute the following command:

analyze_simulation_mismatches -simulation_script runsim

```
// Running script "rnsim work_dft_debug/mentor_default.v_vcdtb.v" from local host to create VCD file
work_dft_debug/mentor_default.v_debug.vcd.
-----
// Step-5 Analyzing 24 internal scan patterns in parallel format.
-----
//   ID  instance (gate_id)  pattern_ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//   ---  ----- (-----)  ----- (-----)  ----- (-----)  ----- (-----)  ----- (-----)
//   1  /ix1286/Y (364)    scan-1      1900       0          1          0  incorrect library cell "and4"
//   2  /ix1278/Y (265)    scan-15     17300       0          1          0  incorrect library cell "and4"
```

Example 4

The following example shows a simulation script for NCVerilog:

```
#!/bin/csh -f
ncsdxfc ../data/good.sdf
#Expect netlist to be compiled, next line commented out
#ncvlog ~/design/netlist.v -v ~/library/lib.v
ncvlog ${1}

#The following two commands create sdfcmdfile for ncelab to load SDF file
echo 'COMPILED_SDF_FILE = "~/design/top.sdf.X",' >! sdfcmdfile
echo "SCOPE = ${2}.design_inst;" >> sdfcmdfile
ncelab worklib.${2}:module -sdf_cmd_file sdfcmdfile
ncsim worklib.${2}:module
```

Example 5

The following example shows how to run a simulation analysis on a remote host.

analyze_simulation_mismatches -auto -host flea01 -simulation_script runsim

analyze_simulation_mismatches

```
// Simulation mismatch source is deleted.
// Running script "runsim mentor_default.v_vcdtb.v" from fleal01 to create VCD file mentor_default.v_debug.vcd.
// Debug 25 internal patterns, CPU time = 0.02 sec
//
// -----
// ID instance (gate_id) pattern-ix time (ns) seq_depth ATPG_val VCD_val mismatch_reason
// -----
// 1 /ix1286/Y (364) scan-1      1900      0        1        0    incorrect library cell "and4"
// 2 /ix1278/Y (265) scan-15     17300     0        1        0    incorrect library cell "and4"
```

Example 6

The following example shows using the **-MAnual** switch:

analyze_simulation_mismatches

```
// -----
// Step-1 Creating Verilog test bench work_dft_debug/mentor_default.v
// with failure file dump.
// -----
// Manual flow is specified by user. Please simulate Verilog test bench
// work_dft_debug/mentor_default.v to create failure file
// work_dft_debug/mentor_default.v.fail.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/mentor_default.v.fail" to generate reduced test bench
// with VCD dump.
```

Upon completion of this step, take the Verilog test bench *work_dft_debug/mentor_default.v* and perform the Verilog simulation to product the failure file *mentor_default.v.fail*.

Then, you issue the following command:

analyze_simulation_mismatches -fail work_dft_debug/mentor_default.v.fail -manual

```
// -----
// Step-1 Loading failure file work_dft_debug/mentor_default.v.fail.
// -----
// Step-2 Creating Verilog test bench
// work_dft_debug/mentor_default.v.fail_vcdtb.v with VCD dump.
// -----
// Manual flow is specified by user. Please simulate Verilog test bench
// work_dft_debug/mentor_default.v.fail_vcdtb.v to create VCD file
// work_dft_debug/mentor_default.v.fail_debug.vcd.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/mentor_default.v.fail -vcd
// work_dft_debug/mentor_default.v.fail_debug.vcd" to complete the
// analysis.
```

After this step, take the Verilog test bench *work_dft_debug/mentor_default.v.fail_vcdtb.v* and perform the Verilog simulation to product the VCD dump file *work_dft_debug/mentor_default.v.fail_debug.vcd*.

Finally, you use the following command to complete the task:

```
ATPG> analyze_simulation_mismatches -fail work_dft_debug/mentor_default.v.fail -vcd work_dft_debug/mentor_default.v.fail_debug.vcd
// -----
// Step-1 Loading failure file work_dft_debug/mentor_default.v.fail.
// -----
// Step-2 Loading VCD file
// work_dft_debug/mentor_default.v.fail_debug.vcd.
// -----
// Step-3 Analyzing 2 internal scan patterns in parallel format.
// -----
// ID instance (gate_id) pattern_ix time (ns) seq_depth ATPG_val VCD_val mismatch_reason
// --- -----
// 1 /u102/ (124) scan-10 2100 0 0 X incorrect loading during the main shifts
```

Related Topics

[report_external_simulator](#)
[report_measure_cycles](#)
[report_mismatch_sources](#)
[write_patterns](#)
[set_external_simulator](#)

analyze_test_points

Context: dft -test_points

Mode: analysis

Specifies the test points analysis and generates a list of test points to be inserted by the tool.

Usage

analyze_test_points

Description

Specifies the test points analysis and generates a list of test points to be inserted by the tool.

If you want to manually add test points before performing the test point analysis, you can do so using the [add_control_points](#) and the [add_observe_points](#) commands. You must do this prior to issuing the `analyze_test_points` command. Use the [delete_test_points](#) command to remove these manually added test points if you do not want the test points to be considered during analysis.

Note

 When you have generated a list of test points with the “`analyze_test_points`” command, then these test points have to be deleted before you can modify the locations where test points are allowed. Any existing test points need to be deleted before issuing the `add_notest_points` and `delete_notest_points` commands.

Note

 Not all gates in the design will be considered by the tool for inserting test points. For various reasons test points will not be inserted on gates in the clock tree or in the scan path, gates in protected blocks, or gates where the user has explicitly requested that no test points should be inserted (via the “`add_notest_points`” command). If there are many locations where test points cannot be inserted the tool may not be able to find the best locations for test points. This may increase the number of test points that is needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20% of the gate-pins the tool will generate a warning. The warning specifies the number of these locations as well as the main reason(s) why test points cannot be inserted at these locations.

There are three distinct purposes for using test points — to improve logic_bist random test coverage, to improve ATPG test coverage or to reduce ATPG pattern counts. The type of test point analysis performed will depend on the setting of the `set_test_point_type` command, the default being logic_bist. For more information, see “[Why You Use Test Points](#)” in the *Tessent Scan and ATPG User’s Manual*.

One typical usage of the test point analysis capability is to identify LBIST test points that will improve fault coverage when applying pseudo-random patterns. This means the analysis is based on the entire population of stuck-at faults within the design. An alternate use of the LBIST test point analysis algorithm is to identify test point locations that may improve fault

coverage when creating and applying deterministic patterns. For example, you could choose to load a fault list that is based on faults that are left undetected (AU, UC, UO, PT or PU) after performing ATPG. Targeting these fault locations will identify test points that improve the probability of detecting these faults, and this might allow the tool to create ATPG patterns that successfully target these faults. For more information, see “[Test Points for Improving the Test Coverage of Deterministic Patterns](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

None

Examples

Example 1

The following example specifies that a maximum number of 50 test points are to be generated for a design that will be tested with a set of 30000 random patterns. Then it performs the test points analysis based on these specifications. After the test points are reported to the screen they are inserted into the netlist:

```
set_system_mode analysis
set_test_point_analysis_options -total_number 50 -pattern_count_target 30000
analyze_test_points
report_test_points
insert_test_logic
```

Example 2

The following example generates 50 test points and performs the test point analysis. The test points are reported to the screen but are then deleted without being inserted as they were only generated for reporting purposes. It then sets the attribute no_control_point to true for block1 and again performs test point analysis. After the test points are reported to the screen they are inserted into the netlist.

```
set_system_mode analysis
set_test_point_analysis_options -total_number 50 -pattern_count_target 30000
analyze_test_points
report_test_points
delete_test_points -all
set_attribute_value block1 -name no_control_point -value true
analyze_test_points
report_test_points
insert_test_logic
```

Example 3

The following is an example of a log file when generating atpg test points to reduce deterministic pattern counts:

```
SETUP> set_test_point_type edt_pattern_count
SETUP> set_system_mode analysis
ANALYSIS> set_test_point_analysis_options -total_number 1000
ANALYSIS> analyze_test_point

//  
// Incremental Test Point Analysis  
//  
//-----  
// TPs 100 = 0 (CP) + 100 (OP),  
// TPs 200 = 0 (CP) + 200 (OP),  
// TPs 300 = 0 (CP) + 300 (OP),  
// TPs 400 = 0 (CP) + 400 (OP),  
// TPs 500 = 0 (CP) + 500 (OP),  
// TPs 600 = 100 (CP) + 500 (OP),  
// TPs 700 = 200 (CP) + 500 (OP),  
// TPs 800 = 300 (CP) + 500 (OP),  
// TPs 900 = 400 (CP) + 500 (OP),  
// TPs 1000 = 500 (CP) + 500 (OP),  
// TPs 1000 = 500 (CP) + 500 (OP),  
//  
// Performing test point optimization...  
//  
// Test point analysis completed: specified number of test points has  
// been identified.  
// Total number of test points 1000  
// Control Points 500  
// Observe Points 500  
// Maximum control point per path 5  
// CPU_time (secs) 51.6  
//
```

Example 4

The following is an example of a log file when generating logic bist test points to improve test coverage:

```
SETUP> set_test_point_type lbist_test_coverage
SETUP> set_system_mode analysis
ANALYSIS> set_test_point_analysis_options -total_number 1000
ANALYSIS> analyze_test_point
```

```
// Identifying locations of x-bounding muxes prior to test point
// analysis.
//
// Test Coverage Report before Test Point Analysis
// -----
// Target number of random patterns      100000
//
// Total Number of Faults                379416
//   Testable Faults                   339781  ( 89.55%)
//     Logic Bist Testable            319101  ( 84.10%)
//     Blocked by xbounding          11597   ( 3.06%)
//     Uncontrollable/Unobservable   9083    ( 2.39%)
//
// Estimated Maximum Test Coverage      93.91%
// Estimated Test Coverage (pre test points) 82.67%
// Estimated Relevant Test Coverage (pre test points) 88.03%
//
//
// Incremental Relevant Test Coverage Report
// -----
// TPs 50 = 49 (CP) + 1 (OP), TC 91.24
// TPs 100 = 93 (CP) + 7 (OP), TC 93.71
// TPs 150 = 141 (CP) + 9 (OP), TC 94.88
// TPs 200 = 172 (CP) + 28 (OP), TC 95.18
// TPs 250 = 212 (CP) + 38 (OP), TC 95.59
// TPs 300 = 262 (CP) + 38 (OP), TC 95.99
// TPs 350 = 308 (CP) + 42 (OP), TC 96.42
// TPs 400 = 352 (CP) + 48 (OP), TC 96.99
// TPs 450 = 392 (CP) + 58 (OP), TC 97.49
// TPs 500 = 440 (CP) + 60 (OP), TC 97.97
// TPs 550 = 475 (CP) + 75 (OP), TC 98.55
// TPs 600 = 500 (CP) + 100 (OP), TC 98.84
//
// Test point analysis completed: target estimated test coverage has been
// achieved.
// Inserted Test Points                 629
//   Control Points                    510
//   Observe Points                   119
//
//
// Test Coverage Report after Test Point Analysis
// -----
// Target number of random patterns      100000
//
// Total Number of Faults                380350
//   Testable Faults                   340715  ( 89.58%)
//     Logic Bist Testable            320035  ( 84.14%)
//     Blocked by xbounding          11597   ( 3.05%)
//     Uncontrollable/Unobservable   9083    ( 2.39%)
//
// Estimated Test Coverage (post test points) 92.99%
// Estimated Relevant Test Coverage (post test points) 99.00%
```

Related Topics

[add_observe_points](#)

analyze_test_points

[delete_test_points](#)
[insert_test_logic](#)
[report_test_points](#)
[set_test_point_analysis_options](#)
[set_test_point_insertion_options](#)
[set_test_point_types](#)
[write_scan_setup](#)
[write_test_point_dofile](#)

analyze_wrapper_cells

Context: dft -scan

Mode: analysis

Identifies shared and dedicated wrapper cells for the primary I/O ports.

Usage

```
analyze_wrapper_cells [-force]
```

Description

Identifies shared and dedicated wrapper cells for the primary I/O ports.

This command launches the identification of the shared wrapper cells for the primary I/O ports that can be controlled with the `set_wrapper_analysis_options` command. The user-specified clocks and scan-related I/O ports are automatically excluded from identification. The wrapper cells identified for the primary I/O ports will be stitched into the wrapper chains. When the `set_wrapper_analysis_options` command was not issued, the `analyze_wrapper_cells` command will perform the default wrapper cells identification similar to issuing the `set_wrapper_analysis_options` command with no arguments.

When the `analyze_wrapper_cells` command is used in conjunction with the `set_dedicated_wrapper_cell_options` command, the wrapper cells analysis will result in registration of the specified primary I/O ports or the primary I/O ports that have failed the shared wrapper cells identification with the dedicated wrapper cells to guarantee better test coverage of the circuit. This means that new scan cells will be added to all specified and failed primary I/O ports during scan insertion, subject to certain design considerations. The dedicated wrapper cells added to the primary I/Os will be stitched into the wrapper chains.

When you use the `set_wrapper_analysis_options` command, with or without the `set_dedicated_wrapper_cell_options` command, the identification includes the identification of existing sequential elements accessible via combinational logic from primary inputs or primary outputs as input/output registration cells (that is, shared wrapper cells). Thus, a shared wrapper cell serves a dual purpose of both functional mode, as well as the control and observe function during test. The advantage of shared wrapper cells is that less logic must be added and the external test mode (ExTest) can test the timing of the actual path.

`Analyze_wrapper_cells` may invalidate data from other analysis stages. This command should be used before any scan chain analysis. If you have already completed scan chain analysis when you issue the `analyze_wrapper_cells` command, it will result in an error. You can use the `-force` switch to override it, in which case the results of scan chain analysis will be reset and you will have to repeat it to recreate scan modes and scan chain families as desired.

The `report_wrapper_cells` command reports information about the identified wrapper cells for each primary I/O port subjected to wrapper cell identification.

Arguments

- **-force**

An optional switch that directs the tool to forcefully issue the analyze_wrapper_cells command. In order to avoid using invalidated data the tool will delete all the scan modes and scan chain families. You will need to recreate the scan modes and scan chain families as needed after you issue the analyze_wrapper_cells command.

Examples

The following example illustrates the use of the analyze_wrapper_cells command:

```
set_dedicated_wrapper_cell_options on -ports out3
analyze_wrapper_cells

Port information and user constraints:
-----
Input ports:      4 total,      1 ignored,      0 excluded,      0 off,      0 on,      3 auto
Output ports:     3 total,      0 ignored,      0 excluded,      0 off,      1 on,      2 auto

Wrapper analysis summary:
-----
1 output port was inferred with a dedicated wrapper cell.
1 input port was inferred with a dedicated wrapper cell.
1 flip-flop was inferred as an output shared wrapper cell.
3 flip-flops were inferred as input shared wrapper cells.
Use report_wrapper_cells for more details.
```

```
report_wrapper_cells -verbose
```

Primary I/O Port	Max Level	Logic Cells	#Shared Wrapper Cells	#Internal Feedback Gates	Wrapper Chain Type	Clock	Inferring Dedicated Wrapper	Inferring Reason Cell							
<hr/>															
'out1'	(O) 0	1	'flop4'		Output	'clk'	No	--							
'out2'	(O) 0	0	/dow_10sffp1_i (new)		Output	'clk'	Yes	Max Logic Level							
'out3'	(O) 0	0	/dow_8sffp1_i (new)		Output	'clk'	Yes	--							
'in2'	(I) 0	1/0	'flop3'		Input	'clk'	No	--							
'in1'	(I) 0	2/0	'flop1'		Input	'clk'	No	--							
			'flop2'		Input	'clk'									
'in3'	(I) 0	0	/diw_18sffp1_i (new)		Input	'clk'	Yes	Max Logic Level							
<hr/>															
The wrapper cells identification has failed for the following PIs and POs.															
These PIs and POs will be I/O Registered unless excluded from the registration.															
<hr/>															
'in3'	Input	Exceeded the limit, 1, on the number of combinational logic levels between this PI and the first level of sequential cells 'out2'													
	Output	Exceeded the limit, 1, on the number of combinational logic levels between this PO and the first level of sequential cells													
<hr/>															
These PIs and POs will be excluded from the I/O Registration unless they were explicitly included.															
<hr/>															
'clk'			Input			Clock	port								
<hr/>															

Related Topics

[report_wrapper_cells](#)
[set_dedicated_wrapper_cell_options](#)
[set_wrapper_analysis_options](#)

analyze_xbounding

Context: dft -scan

Mode: analysis

Performs X-bounding analysis.

Usage

```
analyze_xbounding [-force]
```

Description

Performs X-bounding analysis.

This command is used to identify memory elements (current scan cells or cells that will be converted to scan-in a subsequent run) that might capture an unknown value during Logic BIST. In addition, the command identifies where to place bounding muxes that will prevent these X-sources from reaching the memory elements and corrupting the signature during Logic BIST.

The X-bounding analysis uses the E5 DRC to identify and bound all X sources from reaching scan cells and corrupting the MISR. All primary inputs and internal X source pins (non scan cells, blackbox outputs pins etc.) will be x-bounded.

The X-bounding analysis uses the E9 DRC to identify wire gates that may be driving a net to opposing values (contention). The tool will not add hardware to avoid the contention, however it will add X-bounding logic to prevent observation of the wire gates with the E9 violation.

The X-bounding analysis uses the E10 DRC to detect tristate bus contention. The tool will not add any hardware to avoid the bus contention, however it will add X-bounding logic to prevent observation of the bus with the E10 violation.

The X-bounding analysis uses the E11 DRC to detect a bus that can attain a Z-state. The tool will not add any hardware to prevent the bus from reaching the Z state, however it will add X-bounding logic to prevent observation of the floating bus.

The X-bounding process depends on pin constraints on the primary input pins to indicate them as X sources. These X-constraints are implicitly enforced by this command, but can also come from existing input constraints. When a pin is explicitly constrained to either 0 or 1, it will not be treated as an X source.

The mux and scan cell library models used for X-bounding can be specified using the `cell_type` attribute in the Tessent Cell library or by the [add_cell_models](#) command. If any of the required library cell models are not specified, an error message will be issued.

The `analyze_xbounding` command also marks as notest points the previously observable gate pins that are on the input side of the x-bounding muxes. In addition, an X-source that does not feed into any scannable flops (for example, it simply connects to a primary output pin) will not

be bounded. Any previously observable gate-pins that are driven by that X-source will also be marked as notest points.

Note

 You must issue this command to perform X-bounding analysis. The tool does not automatically perform the analysis when you issue the `insert_test_logic` command.

Analyze_xbounding may invalidate data from other analysis stages. This command should be used before any wrapper analysis and/or and scan chain analysis. If you have already completed any of these steps when you issue the `analyze_xbounding` command, it will result in an error. You can use the `-force` switch to override it, in which case the results of wrapper analysis and scan chain analysis will be reset and you will have to repeat them.

The X-bounding analysis will also bound false and multi-cycle paths. This capability is triggered by reading an SDC file prior to issuing the command. The tool will identify any scannable flops that could capture an X due to a user-defined false or multi-cycle path. The tool will add X-bounding logic that blocks the path at the D-input of the destination of the flop. These flops will “capture” the inverted value of the flop output so they can still trigger transitions at the output during launch-of-capture transition fault testing.

Arguments

- `-force`

An optional switch that directs the tool to forcefully issue the `analyze_xbounding` command. In order to avoid using invalidated data the tool will reset the results of wrapper analysis and will delete all the scan modes and scan chain families before doing X-bounding. It is recommended that you re-run wrapper analysis and set the scan modes and scan chain families as needed before you issue the `analyze_xbounding` command.

Examples

This example transitions from setup to analysis mode, triggers the X-bounding analysis, and then reports the locations of the X-bounding muxes:

```
set_system_mode analysis  
analyze_xbounding  
report_xbounding -verbose
```

Related Topics

[insert_test_logic](#)
[report_xbounding](#)
[read_sdc](#)
[set_xbounding_options](#)
[report_notest_points](#)

annotate_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Adds DFM and RCD data, if available, to existing diagnosis reports.

Usage

annotate_diagnosis *directory_path* [-force]

Description

Adds DFM and RCD data, if available, to existing diagnosis reports.

You can add RCD constants to existing diagnosis reports using the `annotate_diagnosis` command. To annotate a diagnosis report, you must load the same pattern set that you used to run diagnosis and create RCD constants.

If there is already a DFM rule of the same name and type in the report, the tool replaces it. If there are already existing RCD feature statistics in the report, the tool replaces them.

Note

 The `set_diagnosis_options` `-include_dfm_rules` and `-include_rdc_constants` switches do not affect the annotation process. Information is always preserved or added in the annotated diagnosis reports based on the content of the VDB or LDB that is open.

Arguments

- ***directory_path***
A required string that specifies the directory path of a diagnosis report.
- **-force**
An optional switch that specifies to bypass any issues with md5 compatibility that arise.

Examples

Example 1

The following example adds the RCD constants to the `failure_file_directory` diagnosis report.

```
read_flat_model flat_model.gz
read_patterns pattern_file.gz
open_layout ldb_directory
annotate_diagnosis failure_file_directory
```

Example 2

The following example shows how to add RCD constants to existing diagnosis reports. Use the `read_patterns` command to specify the applicable pattern set.

```
# Specify the same pattern set that was used to run diagnosis  
# and create RCD constants  
  
read_patterns /src/patterns.ascii  
  
# Open the LDB  
  
open layout design.dft.ldbannotate_diagnosis small_diags_rcd_gz
```

Related Topics

[delete_dfm](#)

[import_dfm](#)

append_to_collection

Context: unspecified, all contexts

Mode: all modes

Adds one or more objects to a collection referenced by the specified var_name.

Usage

append_to_collection *var_name obj_spec* [-unique]

Description

Adds one or more objects to a collection referenced by the specified var_name.

The append_to_collection command appends all elements in obj_spec to the collection referenced by the variable var_name. If the variable does not exist, the variable is created and made to point to a collection with elements from obj_spec as its value. An error is generated if the variable exists and does not reference a collection.

The result of the append_to_collection command is either a modification of the collection that was initially referenced by var_name or a new collection if the variable did not exist.

The append_to_collection command provides the same semantics as the add_to_collection command but with significant performance improvement. If you are building up a collection in a loop, using the append_to_collection command is more efficient as shown in [Example 2](#).

Arguments

- ***var_name***
A required value that specifies a Tcl variable name. The objects matching obj_spec are added to the collection referenced by this variable.
- ***obj_spec***
A required value that specifies a Tcl list of one or more object names or a collection of one or more objects to append to the collection referenced by var_name. The obj_spec can be an empty collection.
- **-unique**
An optional switch that prevents duplicate objects from being appended to the collection. By default, duplicate objects are not removed.

Examples

Example 1

The following example appends a collection referenced by variable *b* to collection referenced by variable *a*.

```
set a [get_ports a*]
set b [get_ports b*]
append_to_collection a $b
```

Example 2

The following example creates a collection of all nets connected to pins of design instances matching u1/myPrefix*. Note that using the -unique option automatically removes duplicate nets. This example is the same as the example shown for the [add_to_collection](#) command. Using the [append_to_collection](#) command for such an application is recommended and is much more efficient.

```
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
    set inputs [get_pins -of_instance $inst -direction input]
    set outputs [get_pins -of_instance $inst -direction output]
    append_to_collection sum [get_fanins $inputs -stop_on net] -unique
    append_to_collection sum [get_fanouts $outputs -stop_on net] -unique
}
```

Related Topics

[add_to_collection](#)
[compare_collections](#)
[copy_collection](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)

apply_specification_defaults

Context: dft, patterns

Mode: all modes

Automatically applies all defaults specified in the DefaultsSpecification/DftSpecification and/or DefaultsSpecification/PatternsSpecification wrappers to wrappers that were created manually.

Usage

```
apply_specification_defaults [id] [-design_name design_name]  
[-config_object config_object]
```

Description

Automatically applies all defaults specified in the DefaultsSpecification/DftSpecification and/or DefaultsSpecification/PatternsSpecification wrappers to wrappers that were created manually.

You must use this command to apply the defaults under the following circumstances:

- When a DftSpecification or PatternsSpecification has been created manually.
- When a DftSpecification or PatternsSpecification has been modified after the specification has been created and the new defaults should be applied.

You can also use this command when creating a [DftSpecification](#) wrapper in the Configuration Data window. You do not need this command when using the [create_dft_specification](#) command because the [create_dft_specification](#) command automatically applies the specified defaults while creating the DftSpecification wrapper.

Similarly, you can also use this command when creating a [PatternsSpecification](#) wrapper in the Configuration Data window. You do not need this command when using the [create_patterns_specification](#) command because the [create_patterns_specification](#) command automatically applies the specified defaults while creating the PatternsSpecification wrapper.

Note

 Note that you may need to use the `apply_specification_defaults` command to re-apply the defaults when the specification or the “defaults” have been modified *after* the wrappers have been created.

Arguments

- *id*

An optional string that specifies the specification wrapper that is to be processed. As described in the DftSpecification and PatternsSpecification wrapper syntax, the wrapper is identified by the two strings: *design_name* and *id*. If you only have one DftSpecification or PatternsSpecification wrapper present in memory that matches *specification(design_name,*)*, you can omit the *id* from the [process_dft_specification](#) or

process_patterns_specification command. However, if several DftSpecification or PatternsSpecification wrappers matching *specification(design_name,*)* exist in memory, you must specify the appropriate *id* when executing the **process_dft_specification** or **process_patterns_specification** command. Refer to the **-design_name** option to understand how the **design_name** value is obtained.

- **-design_name** *design_name*

An optional switch and string pair that specifies the *design_name* value used to identify the DftSpecification or PatternsSpecification wrapper. The **-design_name** value defaults to the current design if it has been set with the **set_current_design** command. If the current design has not been set, it is extracted from the DftSpecification or PatternsSpecification wrapper existing in memory.

When the *id* is specified, if exactly one wrapper matches “*specification (*,id)*” in memory, the *design_name* is extracted from the *design_name* parameter of the DftSpecification or PatternsSpecification wrapper; if more than one wrapper matches, the **-design_name** must be specified.

When the *id* is not specified, if exactly one wrapper matches “*specification (*,*)*” in memory, the *design_name* is extracted from the *design_name* parameter of the DftSpecification or PatternsSpecification wrapper; if more than one wrapper matches, the **-design_name** must be specified.

- **-config_object** *config_object*

An optional switch and string pair that specifies

An optional switch and string pair that applies the defaults to only one specific DftSpecification or PatternsSpecification wrapper. By default all the specifications in memory that correspond to the selected *id* and “**-design_name** *design_name*” are processed.

Examples

Example 1

The following example calls the **apply_specification_defaults** command on a DftSpecification wrapper that was created manually to get the specified defaults found in the **DefaultsSpecification/DftSpecification** wrapper applied to the DftSpecification wrapper.

apply_specification_defaults

Example 2

The following example uses the **apply_specification_defaults** to reapply defaults to the DftSpecification after the specification has been created.

```

set_context dft -rtl
set_tsdb_out_dir tsdb_outdir_test_read
read_verilog chip.vb
set_current_design chip
set_design_level sub_block
set_system_mode analysis
set_defaults_value \
    DftSpecification/IJTAGNetwork/HostScanInterface/Chip/tck chip_tck
set_defaults_value \
    DftSpecification/IJTAGNetwork/HostScanInterface/Block/tck block_tck

read_config_data -from_string {
    DftSpecification(chip,rtl) {
        IJTAGNetwork {
            HostScanInterface(iJTAG) {
                Tap(TAP1) {
                }
            }
        }
    }
}

apply_specification_defaults

set top [get_config_element DftSpecification]
report_config_data $top
process_dft_specification
extract_icl

```

Related Topics

[create_dft_specification](#)
[create_patterns_specification](#)
[process_dft_specification](#)
[process_patterns_specification](#)
[set_defaults_value](#)

catch_output

Context: unspecified, all contexts

Mode: all modes

Executes the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.

Usage

```
catch_output cmdline [-output output_variable] [-result result_variable] [-tee]
```

Description

Executes the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.

This command returns 0 if the embedded command succeeds, and 1 if it fails.

You can use the -output switch to specify a variable to contain all of the output of the embedded command. You can use the -result switch to specify a variable to contain the Tcl result of the embedded command if the command returns one.

Arguments

- ***cmdline***

A required string enclosed in braces “{ }” that specifies the command and associated arguments whose output you want to capture. The command is typically a report_* command. A return value of 1 indicates the command failed; a return value of 0 indicates the command succeeded.

- **-output *output_variable***

An optional switch and variable name that is set to the value of the entire command output.

- **-result *result_variable***

An optional switch and variable name that specifies to write the Tcl result returned by the embedded command to the *result_variable* variable.

- **-tee**

An optional switch that specifies to write the command output to the transcript and to a logfile.

Examples

Example 1

The following example writes the result of the get_gate_pins command to the variable oVar; the result can be an error message if the command fails. If get_gate_pins fails, catch_output returns 1, and the “if” wrapper commands are executed.

The example also assigns the Tcl result returned by the get_gate_pins command to the rVar variable; the result is a collection if the command succeeds. If get_gate_pins succeeds, the catch_output command returns 0, and the “else” wrapper commands are executed.

The dofile continues even if the get_gate_pins command fails. Nothing is reported to the transcript.

```
if {[catch_output {get_gate_pins foo*} -output oVar -result rVar]} {
    # commands when get_gate_pins fails
} else {
    # commands when get_gate_pins succeeds
}
```

Example 2

The following example uses the catch_output command to capture the output of a tool command in a variable.

```
set retVar [catch_output {report_gate /gate3} -output resultA -tee]
// /gate3 nor02
//   A0      I  /gate2/Y
//   A1      I  /ff2/Q
//   Y       O  /ff5/D  ff4/D  ff3/D  gate6b/E

If { $retVar == 0 } {
    puts $resultA
}

// /gate3 nor02
//   A0      I  /gate2/Y
//   A1      I  /ff2/Q
//   Y       O  /ff5/D  ff4/D  ff3/D  gate6b/E
```

Related Topics

[set_transcript_style](#)

check_design_rules

Context: dft, patterns

Mode: setup

Transitions the tool from setup mode to analysis mode.

Usage

`check_design_rules`

Description

Causes the tool to transition from setup mode to analysis mode.

In RTL context, the design is synthesized in parts before creating the flat model. The DRC rules specific to the current context are then executed. If no DRC with a severity of error fails, the tool enters analysis mode.

If any DRC with a severity of error fails, the tool remains in setup mode, which enables you to analyze the DRC violation and either add more definitions (such as clocks) or make edits to your circuit.

Arguments

None

Examples

The following example shows the results of calling the `check_design_rules` command, which generates a DRC violation. The violation is fixed by defining a missing clock. The `check_design_rules` command is then re-issued and successfully transitions from system mode to analysis mode.

SETUP> `check_design_rules`

check_design_rules

```

// -----
// Begin RTL synthesis.
// -----
// Module 'sub_block1' synthesized.
// Module 'block1' synthesized.
// RTL synthesis completed, synthesized modules=2, Time=1.88 secs.
// -----
// -----
// Begin RTL synthesis.
// -----
// Module 'my_clock_gate' synthesized.
// Module 'my_inv' synthesized.
// RTL synthesis completed, synthesized modules=2, Time=1.94 secs.
// -----
// -----
// Begin RTL synthesis.
// -----
// Module 'my_clock_mux' synthesized.
// RTL synthesis completed, synthesized modules=1, Time=1.84 secs.
// -----
// Warning: Rule FP2 violation occurs 1 times
// Flattening process completed, cell instances=82, gates=843, PIs=5,
// POs=0, CPU time=0.01 sec.
// -----
// Begin circuit learning analyses.
// -----
// Learning completed, CPU time=0.00 sec.
// Error: The memory clock pin 'sub_block1_i1/mem1/CLK' (67.40) source
// tracing stopped at pin 'pll/vco' (9.0).
// Correct the blocking conditions or use the 'add_clocks -period' if
// this is an embedded oscillator or
// add_clocks -reference' if this is an active PLL or clock divider.
// (DFT_C1-1)
// Error: There was 1 DFT_C1 violation (Memory clock not properly sourced
// by a declared clock).

```

SETUP> add_clocks pll/vco -reference pll/ref -freq_mult 10 -freq_divider 2

```
// Warning: Flat model deleted.
```

SETUP> check_design_rules

```

// Warning: Rule FP2 violation occurs 1 times
// Flattening process completed, cell instances=82, gates=844, PIs=6,
// POs=0, CPU time=0.01 sec.
// -----
// Begin circuit learning analyses.
// -----
// Learning completed, CPU time=0.00 sec.

```

ANALYSIS>

check_synthesis

Context: all contexts

Mode: all modes

Checks the status of synthesis that was previously launched by the [run_synthesis](#) command.

Usage

```
check_synthesis [-design_name design_name] [-design_id design_id] [-report_status  
| -get_status_dictionary | -wait | -kill] [-synthesis_output_directory_list directory_list]
```

Description

Checks the status of synthesis run that was previously launched by the [run_synthesis](#) command.

Based on the specified option, you can generate a report, get a status dictionary, or get the pass or fail status.

Arguments

- **-design_name *design_name***

An optional switch and value pair that specifies the design name to which [run_synthesis](#) will be associated with. By default, the name of the *current_design* is used when it is set. If you run the command immediately after running the [process_dft_specification](#) command, you do not need to specify this option.

- **-design_id *design_id***

An optional switch and value pair that specifies the *design_id* to which [run_synthesis](#) will be associated with. By default, the name of the *current_design_id* is used when it is set. If you run the command immediately after running the [process_dft_specification](#) command, you do not need to specify this option.

- **-report_status**

An optional switch that is mutually exclusive with the [-get_status_dictionary](#) and [-wait](#) options. When none of the these options is specified, [-wait](#) is assumed. By default, [-report_status](#) includes all the *design_ids* found in the synthesis directories specified by [-synthesis_output_directory_list](#).

- **-get_status_dictionary**

An optional switch that is mutually exclusive with [-report_status](#) and [-wait](#). When none of these options is specified, [-wait](#) is assumed.

The Tcl dictionary returned with [-get_status_dictionary](#) contains all the information that was gathered to produce the report generated when [-report_status](#) is used instead.

- **-wait**

An optional switch that waits for synthesis task to be completed before returning with a completion message. When running in interactive mode, the command reports a status line

that is updated every second showing at what stage synthesis is current operating, which can be unscheduled, queued, running, pass, or fail, as shown in Example 1.

- **-kill**

An optional switch that will kill the selected running synthesis. The **-kill** switch is mutually exclusive with **-report_status**, **-get_status_dictionary** and **-wait**. If no switch is specified, **-wait** is assumed.

- **-synthesis_output_directory_list directory_list**

An optional switch and value pair that specifies a list of directories that contain the *design_name_design_id.synthesis_synthesis_tool* directories. The default value is *./synthesis_outdir*. You can configure the default value using the “**set_run_synthesis_options -synthesis_output_directory**” command and option

Examples

The following examples show the results of using **check_synthesis** with the available options.

Example 1

SETUP> check_synthesis

```
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
// './synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were
// copied to the current tsdb output directory.
```

Example 2

SETUP> check_synthesis -get_status_dictionary

```
// ./synthesis_outdir {blockB_rtl {status pass date {Mon Aug 24 13:02:38
// PDT 2015} sub_dir blockB_rtl.synthesis_dc_shell log_file
// dc_shell.synthesis_log errors_count 0 errors {}}}
```

Example 3

**SETUP> check_synthesis -synthesis_output_directory_list \
synthesis_outdir**

```
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
// 'synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were copied
// to the current tsdb output directory.
```

Example 4

The following example shows **check_synthesis** usage to report the synthesis status for all *design_ids* in the synthesis output directories named “**check_out_ut1**” and “**check_out_ut2**” for design “chip”:

**SETUP>check_synthesis -design_name chip -report \
-synthesis_output_directory_list [list check_out_ut2 check_out_ut1]**

```
// Synthesis status for check_out_ut2/chip_rtl.synthesis_dc_shell
// =====-
// -----  -----  -----
// Name      Status  Errors  Date
// -----  -----  -----
// chip_rtl  pass    0       Mon Feb  08 09:05:08 PST 2016
// 

// Synthesis status for:
//   (1) check_out_ut1/chip_rtl.synthesis_dc_shell
//   (2) check_out_ut1/chip_rtl2.synthesis_dc_shell
// =====-
// -----  -----  -----
// Name      Status  Errors  Date
// -----  -----  -----
// (1) chip_rtl  fail    3       Thu Feb  04 11:25:49 PST 2016
// (2) chip_rtl2  pass    0       Mon Feb  08 09:05:08 PST 2016
// 
```

Related Topics

[get_run_synthesis_options](#)
[report_run_synthesis_options](#)
[run_synthesis](#)
[set_run_synthesis_options](#)

check_testbench_simulations

Context: dft, patterns

Mode: all

Checks the status of simulations that were previously launched by the run_testbench_simulations command.

Usage

```
check_testbench_simulations
  [-design_name design_name]
  [-design_id design_id]
  [-pattern_id pattern_id]
  [-select pattern_list]
  [-exclude pattern_list]
  [-report_status | -get_status_dictionary | -kill | -wait]
  [-simulation_output_directory_list simulation_output_directory_list]
  [-simulator {questa | vcs | incisive}]
```

Description

Checks the status of simulations that were previously launched by [run_testbench_simulations](#) command.

Based on the specified option, you can generate a report, get a status dictionary, or get the pass or fail status. You can also kill pending simulations or delete patterns simulation directories.

Arguments

- **-design_name *design_name***

An optional switch and value pair that specifies the *design_name* that the patterns to check are associated with. By default, the name of the *current_design* is used when it is set. The *design_name* is part of the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

- **-design_id *design_id***

An optional switch and value pair that specifies the *design_id* to that the patterns to check are associated with. By default, the *design_id* value set using the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

A *design_id* must only use a combination of the following characters:

- Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
- Numeric. For example: “123”.
- Underscore. For example: “A_1”.

- **-pattern_id *pattern_id***

An optional switch and value pair that specifies the *pattern_id* that the patterns to simulate are associated with. By default, the *pattern_id* used by the previously run [process_patterns_specification](#) command is reused automatically; otherwise, the default value of “signoff” is used.

The *pattern_id* is part of the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

- **-select *pattern_list***

An optional switch and value pair that restricts the action on a list of *pattern_name.simulation[_simulation_id]* directories that match at least one of the entries in the *pattern_list*. The name patterns are simple glob patterns with * as the wild card. See [Example 2](#).

- **-exclude *pattern_list***

An optional switch and value pair that excludes a list of *pattern_name.simulation[_simulation_id]* directories that match at least one of the entries in the *pattern_list* from the action of the command. The name patterns are simple glob patterns with * as the wild card. See [Example 3](#).

- **-report_status**

An optional switch that is mutually exclusive with the -get_status_dictionary, -kill and -wait options. When none of those options are specified, -wait is assumed. When the -report_status switch is used, the command creates a report listing the simulation status of each pattern found inside the *design_name_design_id.simulation_pattern_id* directories located inside one of the *simulation_output_directory_list* directories that was simulated, is still simulating, or is waiting to be simulated. The report can be restricted to a subset of the *pattern_name.simulation[_simulation_id]* directories by using the -select or the -exclude options.

The format of the report is illustrated here. The date entry shows the time the current status was achieved. The missing miscompares column is for negative testing where the [run_testbench_simulations](#) -expected_miscompare_count option was used or a *simulator.simulation_log_reference* file was present in the directory.

```
// Simulation status for <simulation_output_directory>
// =====
// -----      -----      -----      -----      -----
// Pattern Name          Status    Unexpected  Missing   Date
// -----      -----      -----      -----      -----
// <pattern_name> (<sim_id>)  unscheduled
// <pattern_name> (<sim_id>)  queued
// <pattern_name> (<sim_id>)  running
// <pattern_name> (<sim_id>)  pass        0          0       Wed Oct  08
//                                         23:38:32 GMT 2014
// <pattern_name> (<sim_id>)  fail        2          0       Wed Oct  08
//                                         23:39:40 GMT 2014
```

After issuing the [run_testbench_simulations](#) command, the “check_testbench_simulation -report_status” command and switch reports the simulation status for the tool operations in the *<pattern_name>.simulation* sub-directory.

For a given simulation, the status normally changes as follows:

- unscheduled — During the compilation phase of the design, all the simulation log files under the *<pattern_name>.simulation* sub-directory are created with an empty content but with the name *<simulator>.simulation_log.unscheduled*. The tool does not queue the *<pattern_name>.simulation_script* at this point because the library of the design is not available until the compilation is completed.
- queue — The tool completes compilation and all simulation log files are renamed from *.unscheduled* to *<simulator>.simulation_log.queued*.
- running — As the simulations start running, the log file is renamed from *.queued* to *<simulator>.simulation_log.running*.

When the simulation completes, the name of the file is renamed from *.running* to *<simulator>.simulation_log*.

- **-get_status_dictionary**

An optional switch that is mutually exclusive with the -report_status, -kill and -wait options. When none of those options are specified, -wait is assumed. When the -get_status_dictionary switch is used, the command returns a Tcl dictionary listing the simulation status of each pattern simulated, simulating, or waiting to be simulated and found inside the *design_name_design_id.simulationpattern_id* directories located inside one of the *simulation_output_directory_list* directories. The dictionary can be restricted to a subset of

the <pattern_name>.simulation[_simulation_id]> directories by using the -select or the -exclude options. The format of the returned dictionary is as follow:

```

<simulation_output_directory> {
    <pattern_name>[.<simulation_id>] {
        status unscheduled | queued | running | pass | fail
        date   <date>
        unexpected_miscompare_count <int>
        unexpected_miscompares {
            miscompare<int> {
                object_name      <object_name>
                step_name        <step_name>
                expected_value   <value>
                simulated_value <value>
            }
        }
        missing_expected_miscompare_count <int>
        missing_expected_miscompares {
            miscompare<int> {
                object_name      <object_name>
                step_name        <step_name>
                expected_value   <value>
                simulated_value <value>
            }
        }
    }
}

```

- **-kill**

An optional switch that is mutually exclusive with the -report_status, -get_status_dictionary, and -wait options. When none of those options are specified, -wait is assumed. When the -kill switch is used, the selected running simulation will be stopped.

- **-wait**

An optional switch that is mutually exclusive with the -report_status, -kill and -get_status_dictionary options. When none of those options are specified, -wait is assumed. When the -wait switch is used, the command waits for all the simulations to be completed before returning with an error message when any of the simulation failed. When running in interactive mode, the command reports a status line that is updated every second showing how many simulations are in the unscheduled, queued, running, pass, or fail state as shown below.

```

check_testbench_simulations
unscheduled <int> queued <int> running <int> pass <int> fail <int>

```

The patterns that are waited on are those simulating or waiting to be simulated, and found inside the *design_name_design_id.simulation_pattern_id* directories located inside one of the *simulation_output_directory_list* directories. The list can be restricted to a subset of the *pattern_name.simulation[_simulation_id]* directories by using the -select or the -exclude options. If you ran the [run_testbench_simulations](#) command using the -generate_scripts_only option or if you issued the check_testbench_simulation -kill option without rerunning the [run_testbench_simulations](#) command, you may have simulation

directories that have no simulation log file and are not scheduled to run. Those should be deleted using the -delete option.

You can use Ctrl-C to stop the waiting and return to the interactive prompt.

- **-simulation_output_directory_list** *simulation_output_directory_list*

An optional switch and value pair that specifies a list of directories that contain the *design_name_design_id.simulation_pattern_id* directories. The default value is *./simulation_outdir*.

- **-simulator** questa | vcs | incisive

An optional switch and value pair to check the status of the simulator used for the simulation that was specified with the [run_testbench_simulations](#) command.

Examples

Example 1

The following example runs the [run_testbench_simulations](#) command to launch the simulations. The [check_testbench_simulation](#) command is then run to monitor the simulations and return an error when the simulations finish and any of them failed. If an error code is trapped by the catch command, the report_status is dumped to the transcript and the tool is terminated with an error code. If no simulation failed, the string “All sims passed” is echoed in the transcript.

```

process_patterns_specification
run_testbench_simulations
if {[catch check_testbench_simulations]} {
    check_testbench_simulations -report_status
    exit -force 1
} else {
    puts "All sims passed"
}

// sub-command: check_testbench_simulations
// Error: 2 out of 4 simulations failed:
//         MemoryBisr_BisrChainAccess with 3 unexpected miscomparisons
//         MemoryBist_P2.FI with 2 missing miscomparisons
// sub-command: check_testbench_simulations -report_status


process_patterns_specification
run_testbench_simulations
if {[catch check_testbench_simulations]} {
    check_testbench_simulations -report_status
    exit -force 1
} else {
    puts "All sims passed"
}

// sub-command: check_testbench_simulations
// Error: 2 out of 4 simulations failed:
//         MemoryBisr_BisrChainAccess with 3 unexpected miscomparisons
//         MemoryBist_P2.FI with 2 missing miscomparisons
// sub-command: check_testbench_simulations -report_status

```

```
// Simulation status for ./simulation_outdir
// =====
// -----  -----
// Pattern Name Status    Unexpected   Missing      Date
//                                     Miscompares Miscompares
// -----  -----
// MemoryBisr_BisrChainAccess fail        3          0      Wed Oct  08 23:38:32 GMT 2014
// MemoryBist_P1                pass       0          0      Wed Oct  08 23:38:35 GMT 2014
// MemoryBist_P2                pass       0          0      Wed Oct  08 23:38:39 GMT 2014
// MemoryBist_P2 (FI)           fail       0          2      Wed Oct  08 23:38:44 GMT 2014
```

Example 2

The following example demonstrates using the -select switch:

```
> check_testbench_simulations -kill -select [list MemoryBisr_BisrChainAccess MemoryBist_P1]
```

Example 3

The following example demonstrates using the -exclude and -select switch:

```
> check_testbench_simulations -kill -exclude [list *Bisr*] -select [list MemoryBis*]
```

Related Topics

[run_testbench_simulations](#)

[report_testbench_simulation_options](#)

close_layout

Context: patterns -scan_diagnosis

Mode: analysis

Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.

Usage

close_layout

Description

Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.

Layout-Aware Diagnosis Flow

SVDB Layout Marker Flow

Releases the Calibre Query Server and closes the layout. If you issue this command, then the tool no longer generates layout markers.

Arguments

None

Related Topics

[create_layout](#)

[open_layout](#)

close_pattern_set

Context: patterns -ijtag

Mode: analysis

Finalizes and closes the currently open pattern_set.

Usage

```
close_pattern_set [-network_end_state {keep | reset | initial}]
```

Description

Finalizes and closes the currently open pattern_set.

If any iRead or iWrite commands remain that have not been processed by an iApply statement, a warning is printed and iApply processes them automatically.

By default, the end state of the ICL network is the one achieved normally by the last iApply statement. When the -network_end_state option is specified with the reset value, the ICL network will be written to such that all network elements are back to their reset state at the end of the pattern_set. When the -network_end_state initial value is used, all network elements are restored to the state they had at the beginning of the pattern_set.

Arguments

- `-network_end_state`

`keep` — No special actions are taken when closing the pattern_set and the state of the ICL network remains in the state reached at the end of the last iApply command. This is the default.

`reset` — Value is used when it is necessary to restore the reset state of the ICL network at the end of the pattern_set. The restoring of the state only applies to registers controlling select lines of ScanMultiplexers; the other registers are not reset. The restoring of the state is performed with iWrite commands to the state control registers. Depending on the ICL network configuration, the restoring of the reset state is often achieved without extra scan operations but some network topology may require that extra scan loads be used.

You use this option when you need to run multiple patterns immediately after this one without doing an iReset between them. Because the reset state is restored at the end of the pattern_sets, you can run them in arbitrary orders.

`initial` — Value used when it is necessary to restore the state of the ICL network at the end of the pattern_set to what it was at the beginning of the pattern_set. The restoring of the state only applies to registers controlling select lines of ScanMultiplexers. The restoring of the state is performed with iWrite commands to the state control registers. Depending on the ICL network configuration, the restoring of the initial state is often achieved without extra scan operations but some network topology may require that extra scan loads be used.

You use this option when you need to run the same pattern_set multiple times in a loop and the pattern_set does not perform an iReset at the beginning. In this case, it is important that the ICL state be identical at the beginning of the first run as it is at the beginning all subsequent runs. The pattern_sets that are typically run in a loop are patterns that perform some kind of polling and the polling routine.

Note

-  To specify a network_end_state for a test_setup procedure, use the “-test_setup_network_end_state” option of the [set_ijtag_retargeting_options](#) command.
-

Return Values

None

Examples

```
open_pattern_set pat1
iCall inst1.mytest
close_pattern_set -network_end_state initial
```

Related Topics

[get_open_pattern_set](#)
[get_pattern_set_list](#)
[open_pattern_set](#)
[report_pattern_sets](#)
[index_collection](#)
[reset_open_pattern_set](#)

close_tsdb

Context: all contexts

Mode: all modes

Makes the contents of a previously opened TSDB directory invisible to the tool.

Usage

```
close_tsdb {directory_path_list | -all} [-silent]
```

Description

Makes the contents of a previously opened TSDB directory invisible to the tool.

The specified TSDB output directory is automatically opened and cannot be closed with this command.

Arguments

- ***directory_path_list***

A required string that specifies an absolute or relative path to a TSDB directory, or a Tcl list containing one or many absolute or relative paths to TSDB directories. You must specify either the ***directory_path_list*** or **-all** option, but you cannot specify both.

- **-all**

A required option that specifies that all previously opened TSDB directories are to be closed. Note that the specified TSDB output directory is automatically opened and remains opened. You must specify either the ***directory_path_list*** or **-all** option, but you cannot specify both.

- **-silent**

An optional switch that suppresses the warning that is normally generated if the specified ***directory_path_list*** is not currently opened or if there are no opened TSDB directories when the **-all** option is used.

Examples

This example closes two TSDB directories with one command invocation.

```
get_tsdb_list
```

```
/home/projects/chipa/tsdbs/corea.tsdb  
/home/projects/chipa/tsdbs/corec.tsdb
```

```
close_tsdb [list ${tsdb_home}/corea.tsdb ${tsdb_home}/coreb.tsdb ]  
get_tsdb_list
```

```
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

[open_tsdb](#)
[get_tsdb_list](#)
[set_tsdb_output_directory](#)
[get_tsdb_output_directory](#)

close_visualizer

Context: unspecified, all contexts

Mode: all modes

Saves display data and closes DFTVisualizer.

Usage

`close_visualizer`

Description

Saves display data and closes DFTVisualizer. Selecting “X” from the window control buttons  performs the same function as `close_visualizer`.

Display data is automatically saved and remains persistent for the current tool session.

Arguments

None

Examples

The following example opens DFTVisualizer, displays the Flat Schematic window, and creates and displays a portion of the design, then closes the viewing session:

```
open_visualizer -display flat_schematic  
add_display_instances i_16_7 -backward -end_point  
close_visualizer
```

Related Topics

[open_visualizer](#)

[write_window_contents](#)

[set_visualizer_preferences](#)

compare_collections

Context: unspecified, all contexts

Mode: all modes

Compares the contents of two collections.

Usage

```
compare_collections collection1 collection2 [-order_dependent]
```

Description

Compares the contents of two collections.

If the same objects are in both collections, the result is 0 (zero). If the objects in the collections are different, the result is a non-zero value. The order of the objects can optionally be considered by using the `-order_dependent` switch.

If two empty collections are compared, the `compare_collections` command considers them identical and returns a result of 0 (zero).

Arguments

- **collection1**
A required string that specifies the base collection for the comparison.
- **collection2**
A required string that specifies the collection to compare with **collection1**.
- `-order_dependent`
An optional switch that causes the order of the objects to be considered when comparing the collections. The collections are considered to be different if the objects are ordered differently, and a non-zero value is returned.

Examples

The following example shows a variety of comparisons.

```
set a [get_ports a*]
{a[3] a[2] a[1] a[0]}

set b [get_ports b*]
{b[3] b[2] b[1] b[0]}

set c [get_ports c*]
{}

#Same objects, same order
compare_collections $a $a
```

```
0  
#Same objects, same order  
compare_collections $a $a -order_dependent  
0  
#Different objects  
compare_collections $a $b  
1  
#Different objects  
compare_collections $a $b -order_dependent  
1  
#Two null collections  
compare_collections $c $c  
0  
#Same objects, different order  
compare_collections [add_to_collection $a $b] [add_to_collection $b $a]  
0  
#Same objects, different order  
compare_collections [add_to_collection $a $b] [add_to_collection $b $a] -order_dependent  
2
```

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[copy_collection](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

compress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Compresses a LDB.

Usage

LDBs created after v2013.3: compress_layout

LDBs created with v2013.3 and before: compress_layout *layout_database_name*

Description

Compresses a LDB.

You must open the LDB using the [open_layout](#) command before performing the compression operation.

See “[Layout Database Compression and Decompression](#)” in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- *layout_database_name*

A required string that specifies the location to and the name of the LDB to be created during the compression operation.

Related Topics

[create_layout](#)

[open_layout](#)

[uncompress_layout](#)

compress_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Compresses patterns in the current test pattern set.

Usage

```
compress_patterns [passes_integer] [-Reset_au] [-MAX_useless_passes integer]
[-MIn_elim_per_pass integer] [-MULTiple_detection [-Order]]
[-MIXed {Essential | Order [-Delta percentage] [-Limit integer]}]
```

Description

Compresses patterns in the current test pattern set.

Note

 You should use `create_patterns` for the smallest pattern set. The `compress_patterns` command is for specialized processes.

The `compress_patterns` command performs static pattern compression on the current test pattern set by repeating fault simulation for the patterns in either reverse or random order and selecting only those patterns required for detection. The *passes_integer* argument specifies the number of pattern compression passes. The first pattern compression pass runs in reverse order and then alternates between random and reverse for additional passes. If you do not specify a *passes_integer* argument, the tool performs only one compression pass.

This is a multiprocessing command for fault simulation. This command supports IDDQ fault models. For more information about multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

If you specify the `-Reset_au` option, then when the tool performs pattern compression, it selects the AU faults for later fault simulation. If during pattern compression these AU faults simulate as possible-detected, the tool labels them as PU (possible-detected—ATPG_untestable), and they receive test coverage credit as possible-detected faults. If the number of pattern compression passes is greater than 1, the tool only performs the resetting of the AU faults for the first pass.

The `compress_patterns` command has a residual memory effect. The initial mode (reverse/random) is not fixed. Instead, it toggles back and forth, starting with the mode last used in the same run. The tool always starts with the reverse order at invocation.

If you include the `-Multiple_detection` switch, the static compression performed by this command will not simulate faults in the posset (PD) fault class (PT and PU subclasses) and as a result, the tool may discard patterns that detect PD faults. Therefore, the test coverage reported after using this command switch may not be accurate if there are PD faults. To ensure you get

the most accurate report of test coverage in this case, re-simulate the patterns using one of the following approaches:

- Enter the “[set_system_mode analysis](#)” command followed by the [simulate_patterns](#) command.
- Enter the [order_patterns](#) command with the -Simulate option.

Arguments

- *passes_integer*

An optional integer that specifies the number of pattern compression passes. The default is 1 (performs only one compression pass).

- -Reset_au

An optional switch that specifies fault simulation of AU faults.

- -MAX_useless_passes *integer*

An optional switch and integer pair that specifies the maximum number of consecutive, useless (no eliminated patterns) passes the tool allows before terminating the pattern compression process. This command option has no effect on the pattern set if the number of passes (*passes_integer* argument) is smaller than the *integer* value of this switch. The default is the *passes_integer* value.

- -MIN_elim_per_pass *integer*

An optional switch and integer pair that specifies the minimum number of eliminated patterns required in a single pass to continue the pattern compression process. If you specify this switch, you must enter a value greater than 0.

- -Multiple_detection

An optional switch that specifies for the tool to use a multiple detection based algorithm to carry out static pattern compaction. When you use this switch, the tool runs only one pass of compaction and discards patterns that detect only posdet (PD) faults.

- -Order

An optional switch used with the -Multiple_detection switch to specify pattern ordering at the same time as static pattern compaction.

- -MIXed Essential | Order

An optional switch and string pair that specifies to use a mixed compaction technique to compact the current internal pattern set. If you use the “Essential” option, the tool considers essential faults, those faults detected only once in the pattern set, as candidates for dynamic compaction. This option ensures that the patterns used to detect the essential faults are not dropped during pattern compression to keep the reached coverage.

If you use the “Order” option, the tool selects fault candidates for dynamic compaction based on pattern set ordering.

Note

 Using the “-Mixed Order” switch turns off pattern classification until you issue a “set_pattern_classification On” command.

Mixed compaction can reduce the size of the current internal pattern set in most situations, but Mentor Graphics cannot guarantee that in every case the size of the pattern set after the tool performs mixed compaction will be less than the size of the pattern set before mixed compaction. Therefore, be sure to save the current internal pattern set before carrying out mixed compaction.

- **-Limit integer**

An optional switch and integer pair that modifies the behavior of the -Mixed option. The -Limit switch specifies the maximum number of faults that the tool will attempt unsuccessfully to merge with the target fault pattern when dynamic compaction is involved. Without the -Limit switch, -Mixed causes the tool to use an adaptive algorithm to determine this number.

- **-Delta percentage**

An optional switch and real number pair (between 0 and 100) that modifies the behavior of the -Mixed Order option. Normally, -Mixed Order selects fault candidates for dynamic compaction based on pattern set ordering. The -Delta switch causes the tool to consider as candidates for dynamic compaction, the specified percentage of the total number of faults instead of using pattern set ordering. The default percentage is 2.

Examples

The following example compresses the generated test pattern set with two passes; the first pass is by reverse order and the second pass is by random order:

```
set_system_mode analysis
create_patterns
compress_patterns 2
```

Related Topics

[add_processors](#)
[create_patterns](#)

copy_collection

Context: unspecified, all contexts

Mode: all modes

Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.

Usage

`copy_collection collection`

Description

Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.

The only time you need to copy a collection is when you want to append to one copy while leaving the original copy unchanged. Copying a collection is equivalent to using `add_to_collection` with an empty collection as the second collection.

Arguments

- ***collection***

A required string that specifies the collection to be copied.

Examples

The following example shows the results of copying a collection and then appending to the copy. Notice how *a2* is modified by `append_to_collection`, but *a1* remains unaffected. The second reference to *a2* called *a3* sees the updated collection because it is simply a second reference to the same collection referenced by *a2*.

```
set a1 [get_ports a*]
{a[3] a[2] a[1] a[0]}

set a2 [copy_collection $a1]
{a[3] a[2] a[1] a[0]}

set a3 $a2

append_to_collection a2 [get_ports b[0]]
{a[3] a[2] a[1] a[0] b[0]}

puts [get_name_list $a1]
{a[3]} {a[2]} {a[1]} {a[0]}

puts [get_name_list $a2]
{a[3]} {a[2]} {a[1]} {a[0]} {b[0]}
```

```
puts [get_name_list $a3]  
      {a[3]} {a[2]} {a[1]} {a[0]}
```

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[compare_collections](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

copy_module

Context: all contexts

Mode: insertion

Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a [create_instance](#) and [replace_instances](#).

Usage

```
copy_module obj_spec -new_name module_name [-silent]
```

Description

Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a [create_instance](#) and [replace_instances](#).

The `copy_module` command returns a collection with one module object corresponding to the newly created module.

Arguments

- ***obj_spec***

A required value that specifies a module name or a collection containing a single module object as returned by the [get_modules](#) command.

- **-new_name *module_name***

A required switch and value pair that specify the name of the new module once it has been copied. If *module_name* already exists, the module is copied and “_integer” is appended to the end of the *module_name* where integer increments as each new module of the same name is added.

- **-silent**

An optional switch that suppresses the error that is normally generated if *obj_spec* does not point to a single module that exists. A null collection is returned when an error was suppressed. This option also suppresses the error that is generated if the specified *module_name* already exists.

Examples

Example 1

The following example creates a copy of module ModA to ModB. ModB is identical to ModA and can be instantiated into the design.

```
copy_module ModA -new_name ModB  
{ModB}
```

Example 2

The following example introspects the module name of an instance to create a copy.

```
copy_module [get_att_value u1/u2 -name module_name] -new_name ModB  
{ModB}
```

Related Topics

[create_instance](#)
[get_nets](#)
[get_icl_ports](#)
[create_module](#)
[get_icl_pins](#)
[get_common_parent_instance](#)
[replace_instances](#)

copy_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist.

Copies the simulation values and forces from one simulation context to another.

Usage

```
copy_simulation_context [-from source_context_name] [-to target_context_name]
```

Description

Copies the simulation values and forces from one simulation context to another.

The `copy_simulation_context` command copies the simulation values and forces from any predefined or user-defined simulation context to any user-defined simulation context.

Note that you must specify either the `-to` or the `-from` switch, or specify both switches.

Arguments

- `-from source_context_name`

An optional switch and string pair that specifies the simulation context from which to copy the simulation values and forces. The `source_context_name` can be predefined (listed in [Table 6-7](#) on page 2023) or user-defined. If you do not use this switch and you do supply the `-to` switch, the command uses the current simulation context as the `source_context`.

- `-to target_context_name`

An optional switch and string pair that specifies the simulation context to which to copy the simulation values and forces. The `target_context_name` can be any user-defined context name that already exists and cannot be a predefined context name (listed in [Table 6-7](#) on page 2023). If you do not use this switch and you do use the `-from` switch, the command uses the current simulation context as the `target_context`.

Examples

The following examples show two ways to copy the simulation values of the predefined simulation context, `stable_load_unload`, to the user-defined simulation context, `myContext`:

```
SETUP> set_current_simulation_context myContext  
SETUP> copy_simulation_context -from stable_load_unload
```

You can accomplish the same as the above with a single command:

```
SETUP> copy_simulation_context -from stable_load_unload -to myContext
```

Related Topics

[add_simulation_context](#)

[get_simulation_value_list](#)
[add_simulation_forces](#)
[report_simulation_contexts](#)
[delete_simulation_contexts](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[simulate_clock_pulses](#)
[get_simulation_context_list](#)
[simulate_forces](#)

create_bisr_segment_order_file

Context: all contexts

Mode: all modes

Creates the BisrSegmentOrderSpecification configuration data wrapper and extracts the list and ordering of BISR chain segments.

Usage

```
create_bisr_segment_order_file [design_id] [-replace]
```

Description

This command extracts the list of repairable memory instances by inspecting the ICL file and performing design introspection of all memories that have the [set_memory_instance_options](#) -use_in_memory_bisr_dft_specification attribute set. The ScanInPort pins of BISR chains found on child instances within included ICL modules will also be extracted.

When a DEF/PDEF file is available, the BISR chain is extracted and the ordering is optimized based on the coordinates of the memory instances. If the DEF/PDEF is specified for a block instance, the ordering is optimized based on the block's BISR_SI pin coordinates. If a block's BISR_SI pin does not have the coordinates specified, but the parent block does, those coordinates will be used for ordering optimization. If DEF/PDEF information is not available, the BISR chain is alphabetically sorted by memory instance names.

The BISR Segment Order File can be edited and the BISR chain can be re-ordered during the insertion phase of the design flow. Any BISR registers that do not appear in the BISR Segment Order File will not have a BISR module generated and will have its repair ports untouched. For further information on assigning and controlling BISR chains, refer to the “[Inserting BISR Chains in a Block](#)” topic in the *Tessent MemoryBIST User’s Manual For Use With Tessent Shell* manual.

The format of the BISR Segment Order File containing the BisrSegmentOrderSpecification wrapper is shown in [Figure 10-53](#). The file containing the BisrSegmentOrderSpecification wrapper is referenced by the DftSpecification/[MemoryBisr](#) bisr_segment_order_file :file_name property.

Arguments

- *design_id*

An optional string that identifies the design. The default is the current design.

- -replace

An optional switch that specifies the existing wrapper is to be replaced. If the command is run and finds a BISR Segment Order File already exists for the specified *design_id*, an error is generated informing the user to use -replace.

create_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.

Usage

```
create_capture_procedures [-Name capture_procedure_name]
  {-PAtterns [Internal | External] [pattern_index...]}
  {{-Clock_sequence clock_sequence | -SAME_clocks_pulse
    {SIngle_clock_per_procedure | GRoup_noninteracting_clocks_per_procedure}
    [pulse_times]}} [-FORce_pi [cycle_number...]] [-MAsk_po]
  [-SLow_cycles {cycle_number...}]] [-Load_cycles {cycle_number...}]
  [[-COndition pinpathname value [cycle_number...]]]...
  [[-TImeplate [Fast | Slow] timeplate_name]...]
  [{-EXT_Clock_sequence clock_sequence | -EXT_Same_clock_pulse external_clock_name
    pulse_times} [-EXT_Timeplate timeplate_name]]} [{>|>>} file_pathname]
```

Description

Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.

The `create_capture_procedures` command is used before or after pattern generation. This command creates the procedures in the internal memory only. You can save the procedure to a test procedure file using the [write_procfile](#) command.

For information on using named capture procedures, see “[At-Speed Test With Named Capture Procedures](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **-Name** *capture_procedure_name*

An optional switch and string pair that specifies the name of the capture procedure to create. If there is already a procedure with this name, a suffix is added to the name specified. If multiple named capture procedures are created, the specified *capture_procedure_name* is used as the prefix of each name and a suffix is added. If -Name is not specified, the default name is *user_capture_proc1*, *user_capture_proc2*, and so on.

- **-PAtterns** [Internal | External] [*pattern_index...*]

A switch that specifies which patterns to extract clock sequences from for the named capture procedures. The following options are available:

Internal | External — An optional literal that extracts clock sequences from internal or external patterns. The default is internal if not specified.

[*pattern_index...*] — An optional, repeatable string that extracts clock sequences from the patterns with the specified index. The pattern index starts at 1. By default, the tool extracts clock sequences from all the patterns of the internal or external pattern set if a pattern index is not specified.

- **-Clock_sequence *clock_sequence***

A switch and string pair that specifies the clock sequence for the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or more clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The clock is designated by the gate identification number or pin pathname. For example, (c1,c2) (c1) (c2,c3) represents a clock sequence with three cycles. In the first cycle, clock c1 and c2 are pulsed; in the second cycle, clock c1 is pulsed; in the third cycle, clock c2 and c3 are pulsed.

- **-SAME_clocks_pulse {SIngle_clock_per_procedure
| Group_noninteracting_clocks_per_procedure} [*pulse_times*]**

A switch, literal, and optional value that automatically generate intra clock domain named capture procedures.

SIngle_clock_per_procedure — A literal that pulses a single clock in every pulse cycle. A named capture procedure is automatically created for every clock in the design that is not constrained to its offstate value. Also, all clocks defined as its equivalence are pulsed when a single clock is pulsed in every cycle.

Group_noninteracting_clocks_per_procedure — A literal pulses a clock group, which consists of a group of non-interacting clocks in every pulse cycle. Each clock group consists of a maximal set of non-interacting clocks. Once clocks are classified into groups, a named capture procedure is created for every clock group. (To understand the difference between non-interacting and interacting clocks, see the description of interacting clocks in the “domain_clock” and “-any_interaction” options in the [set_clock_restriction](#) command.)

pulse_times — An optional value that specifies the number of cycles that pulse clock(s). By default, *pulse_times* is 2 if not specified.

- **-FORce_pi [*cycle_number...*]**

An optional switch and repeatable integer that force PI values in the specified cycle(s). If *cycle_number* is not specified, all the cycles contain the force_pi statement. By default, only the first cycle contains the force_pi statement. The cycle number starts at 1.

- **-MAsk_po**

An optional switch that specifies the last cycle contains the no measure_po statement. By default, the last cycle contains measure_po to indicate the time when the PO measurement happens.

- **-Slow_cycles {*cycle_number...*}**

An optional switch and repeatable integer that specify the slow cycles of the procedure.

- **-Load_cycles {cycle_number...}**
An optional switch and repeatable integer that specify where the scan load happens before the capture cycles of the procedure. By default, only the single scan load happens before the first cycle.
- **-Condition pinpathname value [cycle_number]**
An optional switch, pin pathname, integer, and cycle integer that specify a condition of the scan cells in the named capture procedure. If *cycle_number* is not specified, the condition is a global condition; otherwise, the condition is a local condition associated with the specified cycle only.
- **-TImeplate [Fast | Slow] timeplate_name**
An optional switch, literal, and string that specify the timeplate for the fast or slow cycles of the named capture procedure. By default, the specified timeplate applies to the fast cycles of the named capture procedure.
If this optional switch is not specified, the default timeplate is used for the named capture procedures. You can define the default timeplate by specifying the Set Default_timeplate statement in the procedure file.
- **-EXT_Clock_sequence clock_sequence | -EXT_Same_clock_pulse external_clock pulse_times [-EXT_Timeplate timeplate_name]**
An optional switch, string, and value that automatically create the external mode of the named capture procedures. This switch should not be used together with the **-Patterns** switch.
 - EXT_Clock_sequence clock_sequence** — An optional switch and string pair that specifies the clock sequence for the external mode of the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or multiple clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The designated clock(s) must be external clock(s).
 - EXT_Same_clock_pulse external_clock pulse_times** — An optional switch, string, and value that specify to pulse an external clock in every cycle in the external mode. You can specify the external clock by its gate identification or pin pathname. *pulse_times* specifies the number of cycles that pulse the external clock.
 - EXT_Timeplate timeplate_name** — An optional switch and string that specifies the timeplate to use for the external mode of the named capture procedure. The default timeplate is used if this optional switch is not specified.
- **>file.pathname**
An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file.pathname*.
- **>>file.pathname**
An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file.pathname*.

Examples

Example 1

The following example creates named capture procedures based on the clock sequences extracted from internal patterns.

```
set_system_mode analysis
create_patterns
create_capture_procedures -name my -patterns internal
```

Example 2

The following example creates named capture procedures based on the clock sequences extracted from the 1st, 12th, and 20th patterns in the external pattern set.

```
read_patterns pattern_file
create_capture_procedures -name my -patterns external 1 12 20
```

Example 3

The following example creates a named capture procedure of two cycles for every non-constrained clock in the design.

```
create_capture_procedures -same_clocks single 2
```

Example 4

The following example creates a named capture procedure with a user-defined clock sequence. The defined clock sequence has four clock cycles. In cycle 1, clock1 is pulsed. In cycle 2, clock1 and clock2 are pulsed simultaneously. In cycle 3, clock1 and clock3 are pulsed simultaneously. In cycle 4, clock1 is pulsed and PO is measured. In this example, all cycles are at speed cycles.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1)
```

Example 5

The following example creates a named capture procedure using the same clock sequence defined in the previous example. In the first cycle, the scan cell needs to satisfy the condition of having the loading value of 1.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1) -condition /inst_core/inst_iacc/inst_reg 1 1
```

Related Topics

[delete_capture_procedures](#)
[report_capture_procedures](#)
[set_capture_procedures](#)
[write_procfile](#)

create_connections

Context: all contexts

Mode: insertion

Connects pin, net, or port objects.

Usage

Usage 1: 1-to-1 pairing connections

```
create_connections object_spec1 object_spec2  
[-net_name net_name] [-net_uniquification_suffix net_uniquification_suffix] [-silent]
```

Usage 2: 1-to-Many broadcast connections

```
create_connections object_spec1 object_spec2  
[-net_name net_name] [-net_uniquification_suffix net_uniquification_suffix]  
[-treat_common_ancestor_as_unique on | off] [-silent]
```

Usage 3: Many-to-1 constant connections

```
create_connections object_spec1 -constant {1 | 0}  
[-net_name net_name] [-net_uniquification_suffix net_uniquification_suffix] [-silent]
```

Description

Connects pin, net, or port objects.

You can connect any of the object types pins, nets, or ports as described here:

Usage 1: When *object_spec1* and *object_spec2* contain the same number of elements, the tool pairs the elements from the two lists in the order of the two lists and creates a connection between each pair. Only one driver element per pair is allowed.

Usage 2: When *object_spec1* has one element, and *object_spec2* has one or more elements, *object_spec2* cannot include any driver elements, such as an input port or an output pin.

Usage 3: When you specify *-constant*, you cannot specify *object_spec2* and *object_spec1* cannot contain any driver element, such as an input port or an output pin.

When the *-net_name* option is not specified, the net and pin name used to create the connection is automatically inferred from the name of the elements within *object_spec1* using the following algorithm:

- If the element in *object_spec1* is a pin, the tool uses the *leaf_pin_name* unless it has fewer than three characters; if it has fewer than three characters, the tool uses the *leaf_instance_name* of the pin concatenated with the *leaf_name* of the pin and separated by an underscore.
- If the element in *object_spec1* is a net or a port, the tool uses the *leaf_name* irrespective of the number of character it has.

You can override the default net name by using the `-net_name` switch and specifying the `net_name` option to be used to create the intermediate nets and pins. Regardless of whether the default or a user-specified net name is specified, the tool checks for net conflicts and uniquifies the newly created net to avoid accidental shorts. The `uniquify` operation is performed using the `net_uniquification_string` suffix specified with the `set_insertion_options` command.

The two endpoints of each connection may be in different parts of the design hierarchy. The tool creates all of the intermediate pins needed to create the connection through the hierarchy. The tool automatically reuses existing pins that already make up part of the requested connection.

If you connect more than one pin within a given instance to a common source outside the instance, the tool creates only a single pin on the instance interface and reuses that same pin for all of the desired connections specified within the instance.

Arguments

- ***object_spec1 object_spec2***

A required value that specify a set of endpoints of the connections to be created. The value of the `object_spec` argument can be a Tcl list of one or more pin, port, or net names; or a collection of one or more pin, port, or net objects.

The size of `object_spec1` must either be one or equal to the size of the `object_spec2` when `object_spec2` is specified. The `object_spec1` argument cannot include any driver elements when you specify the `-constant` option. At most, one driver can exist in each connection pair when you specify both `object_spec1` and `object_spec2`.

If an object does not exist within one of the `object_spec` and you specify the `-silent` option, the tool ignores the object and skips the connection request. If the object does not exist and you do *not* specify the `-silent` option, the tool generates an error.

- **`-constant 1 | 0`**

A switch and literal pair that specifies a constant value to connect to all elements within `object_spec1`. The `-constant` switch is mutually exclusive with the `object_spec2` argument.

When this option is specified, the `object_spec1` argument cannot contain any driver elements such as input ports or output pins. In Verilog, a 1'b0 or 1'b1 connection is used as the connection to the pins, and a buf primitive or an assign statement with its input connected to 1'b0 or 1'b1 is used as the connection to output ports or to nets.

- **`-net_name net_name`**

An optional argument that specifies the name of the pin created on `object_spec1` or `object_spec2` to route the connection. If `-net_name` is not specified, the tool uses the leaf name of `object_spec1` to name the pin. If `object_spec1` is a pin and its leaf name has fewer than three characters, the tool uses the leaf instance name of the base name of the pin concatenated with the leaf pin name and separated by an underscore.

- **-net_uniquification_suffix *net_uniquification_suffix***

An optional argument that defines the string that Tessent Shell uses to uniquify a net when the net that is trying to be created already exists in the design. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the net name does not already exist. For example, specifying -net_uniquification_suffix of _TS# will append the string _TS# to the net where # is replaced by the first integer above 0, making the net name unique. If net A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new net with default name A and would use B_TS2 if it tried to create a new net with default name B.

When unspecified, the suffix defaults to the value set by the `set_insertion_options -net_uniquification_suffix` command.

- **-treat_common_ancestor_as_unique *on | off***

An optional switch and literal pair that specifies whether the `create_connection` command treats the common ancestor module of a one-to-many connection as unique. By default, the tool does not lift the connections as if the common ancestor is unique (that is, instantiated only once). This results in the same one-to-many connections being made in all instances of the common ancestor module. This is the setting used when instantiating the instruments as part of the `process_dft_specification` command.

When set to off, the connection is lifted above the common ancestor if the common ancestor is non-unique and the connection does not include all repeated instances of the destinations. This allows the remaining destinations to be connected to a different source. This functionality is only available when auto-uniquify is off.

When unspecified, the setting defaults to the value set in the `set_insertion_options -treat_common_ancestor_as_unique` command.

- **-silent**

An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages. Connections with at least one non-existent object are skipped.

Examples

Example 1

The following example illustrates Usage 1 and creates a connection between three scalar ports and a 3-bit bus pin. The pin name created on u2 to route the connection is u2_a[2:0]. This pin name is constructed by concatenating the `leaf_instance_name` of u1/u2/a with the `leaf_name` of pin a with an underscore, because it has fewer than three characters.

```
create_connections u1/u2/a[2:0] {a b c}
```

Example 2

The following example illustrates Usage 2 and creates three connections between port IN1 and pins blockA/modB/u1/a, blockA/modB/u2/a, and blockA/modC/u1/a. The command creates a single net called myname on u1, u1/u1 to make all three connections.

[create_connections](#)

```
create_connections IN1 {blockA/modB/u1/a blockA/modB/u2/a blockA/modC/u1/a} \
-net_name myname
```

Example 3

The following example illustrates Usage 3 and connects all pins matching the name ABC*/TE to a constant 0 value.

```
create_connections [get_pins ABC*/TE -hier] -constant 0
```

Example 4

The following example illustrates Usage 2 and shows a one-to-many connection involving Verilog generate loop statements. The ltest_to_en output on a module outside the generate loop is broadcasted to the LV_TM inputs on instances inside 2 different generate loops.

```
create_connections {
    TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
    TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
    TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
    TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
} {
    param_parent_genloop2_l/child_genloop[0].memory_interface_inst/LV_TM
    param_parent_genloop2_l/child_genloop[1].memory_interface_inst/LV_TM
    param_parent_genloop3_l/child_genloop[0].memory/memory_interface_inst/LV_TM
    param_parent_genloop3_l/child_genloop[1].memory/memory_interface_inst/LV_TM
}
```

Related Topics

[delete_connections](#)
[get_nets](#)
[get_icl_pins](#)
[get_icl_ports](#)
[move_connections](#)

create_dft_specification

Context: dft (with no sub context)

Mode: analysis

Creates the “DftSpecification(*design_name,id*)” configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.

Usage

```
create_dft_specification [-existing_ijtag_host_scan_in host_scan_in_design_pin_spec]
[-existing_master_tap_scan_out tap_client_scan_out_design_pin_spec]
[-existing_bscan_host_scan_in bscan_host_scan_in_design_pin_spec]
[-active_high_compliance_enables enable_port_name ...]
[-active_low_compliance_enables enable_port_name ...]
[-sri_sib_list sri_sib_list]
[-sti_sib_list sti_sib_list]
[-replace]
```

Description

Creates the “DftSpecification(*design_name,id*)” configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.

This command finds Ijtag instances in the design that need to be connected to the IjtagNetwork and automatically creates the IjtagNetwork wrapper to connect them. The command supports multiple DFT insertion passes into any given module. The tool excludes elements seen to already be connected to the network from the current specification. An Ijtag instance is automatically seen by the command if its ICL file was loaded and the ICL module was matched to the design module during elaboration. See “[set_design_sources -format](#)” and the [set_module_matching_options](#) command descriptions for more details about this process. Use [report_ijtag_instances](#) or [get_ijtag_instances](#) to see which instances have a matched ICL module.

This command also creates the necessary IJTAG network to control created memory BIST and memory BISR controllers as well as providing a boundary scan host port to BoundaryScan. To create a DFT Specification for any of these three features, you must enable them in setup mode using the [set_dft_specification_requirements](#) command prior to issuing the [check_design_rules](#) command. Refer to the BoundaryScan sections, if you are not familiar with these flows. Even though they are documented in separate sections, you can insert all instrument types in one insertion pass or in successive passes. The tool supports both methods and is simply controlled by what options you use with the [set_dft_specification_requirements](#) command prior to issuing the [check_design_rules](#) command.

Note

 If you have used [add_dft_signals](#) scan_en to define top level ports as DFT signals of type scan_en, it will be automatically added to the [AdvancedOptions](#)/ConstantPortSettings wrapper such that the scan enable signals are automatically asserted off during the pattern generation. If you have other scan enable ports not identified with the add_dft_signals, you need to add them yourself in the [create PatternsSpecification](#).

Note

 If you wish to perform boundary scan insertion and TAP insertion pointing directly to internal pins instead of top-level ports, refer to the instructions in [Example 2](#) of the [IjtagNetwork](#) wrapper section.

How the IjtagNetwork specification is created

The [set_design_level](#) command is used to define the target usage level of the current design. When the level is set to “physical_block” or “sub_block”, the IJTAG interface created by the [create_dft_specification](#) command is a client IJTAG interface. When the level is set to “chip”, it is a client TAP interface. The TAP interface port names can be set by [create_dft_specification](#) if the ports are identified with the [Port](#) built-in attribute “[function](#)” set to tdi, tdo, tms, tck, or trst. If a port function is not set, then [create_dft_specification](#) will revert to the [DefaultsSpecification](#) value or the default value of the property.

For the first dft insertion pass containing any IJTAG component, a root level IJTAG ring is created with one or two Sibs in series. The two sibs are called Sib(sti) and Sib(sri) and are inserted on demand when needed by the current insertion pass if it was not already inserted by a prior insertion pass. The Sib(sti) provides access to the IJTAG network for all instruments in the current design that are to be scan tested along with the rest of the functional logic by the logictest modes such as EDT or logicBIST. The acronym “sti” refers to “Scan Tested Instrument”. As shown in [Figure 10-11](#), that Sib provides isolation between its host port and its clients. The Sib(sri) is used to provide access to the IJTAG network for child physical blocks, and for logictest instruments and Tdrs used to configure the logictest modes. The acronym “sri” refers to “Scan Resource Instrument”. These IJTAG nodes must remain active and undisturbed during the logictest modes, and are excluded from the logictest scan chains.

An ICL module is identified as a Scan Resource Instrument when it has the ICL module attribute called [keep_active_during_scan_test](#) set to “true”, which is the default, and that the instrument is treated as a Scan Resource Instrument and placed under the Sib(sri). If you have an instrument which you want to be tested like any other functional logic during scan test, you must specify the [keep_active_during_scan_test](#) attribute to “false” in its ICL module.

At the chip level, a TAP controller is inserted on the first insertion pass unless the [-existing_ijtag_host_scan_in](#) option of the [create_dft_specification](#) command has been used to reference a design pin. The design pin must be on an instance having an associated ICL module and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. Those different scenarios are illustrated in the [Example 1](#) section below.

Any instance of a module having a corresponding ICL module will, by default, be inserted into the IJtagNetwork wrapper of the DftSpecification unless it is seen as being already connected to the network. The [set_ijtag_instance_options](#) command or the [tessent_use_in_dft_specification](#) `icl_module` attribute can be used to change the default behavior.

Different ICL elements (ScanInterfaces, DataInPorts, and DataOutPorts) seen in the circuit are handled by this command as described here:

- Client ScanInterface — A Client ScanInterface is referenced by a [DesignInstance](#) wrapper when it is to be inserted into the network as part of the current DftSpecification. It is not referenced by the current IJtagNetwork when the ScanInPort associated to the Client ScanInterface has its attribute “`has_functional_source`” set to true. You can use the [set_ijtag_instance_options](#) command to force its inclusion or exclusion independent of the value of the `has_functional_source` attribute.

If the ScanInterface is on an instance that is defined as a physical block (as defined by the “`tessent_is_physical_module`” attribute of its ICL module), it will be inserted below the Sib(sri).

If the ScanInterface is on an instance that is not defined as a physical block (as defined by the “`tessent_is_physical_module`” attribute of its ICL module), it will be inserted behind the Sib(sti) or Sib(sri) based on the value of the “`keep_active_during_scan_test`” attribute on its ICL module.

If you have read in UPF or CPF descriptions using the [read_upf](#) or [read_cpf](#) commands, the information will be used to make sure that groups of DesignInstances associated to different power domains are isolated by a SIB. This is done to avoid daisy chaining them with elements from a different power domain and running into issues when accessing the elements in a given power domain while the other power domains are off.

- DataInPorts — A DataInPort is referenced by a connection property inside the Tdr/ DataOutPorts wrapper when it is to be controlled by the network. The DataInPort is not referenced by the current IJtagNetwork when there is already a DataOutPort or a primary input port in its fanin or the DataInPort has the `icl_port` attribute “`tessent_use_in_dft_specification`” set to “false”. If the `tessent_use_in_dft_specification` attribute is set to “auto_no_pi”, the port will be sourced by a TDR port even if it is sourced by a primary input port. You use this value on special ports like the reset port of a PLL or clock divider to make sure it will be locally controllable by IJTAG. This concept is further explained in the [ProcedureStep](#) wrapper section of the [PatternsSpecification](#) chapter. It contains a complete example of a clock divider re-used by memory BIST and initialized by IJTAG. If the DataInPort has a functional source, a multiplexer will be inserted to allow controlling it by the IJtagNetwork in test mode without disturbing its functional source its mission mode. You can use the [set_ijtag_instance_options](#) command to force its inclusion or exclusion independent of whether it has a DataOutPort in its fanin.

If the instance on which the DataInPort exists has the `keep_active_during_scan_test` attribute defined and set to “true”, the Tdr sourcing the DataInPort will be inserted

behind the Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the `keep_active_during_scan_test` attribute is “false”, then the Tdr sourcing the DataInPort will be inserted behind the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

The reset value of the TDR bits will be determined first by the constant tie value of the design pin, or second by the ICL attribute “`default_load_value`” of the ICL description of that port if the design pin is not tied. The reset value will default to 0 if none of these conditions are found.

You can use the `tessent_enable_group` attribute to group the DataInPorts into arbitrary groups. One Tdr will be inserted per enable group enabling the multiplexers to be switched to the IJTAG side independently for each group and thus allowing taking control of one group of ports while leaving the others under functional control.

You can also use the `tessent_common_tdr_source` attribute to drive the DataInPorts from a single TDR bit. Every unique source name creates a new DataOutPort on a single TDR across all design instances. If the `tessent_enable_group` is also specified, a new TDR is created for each group to drive all the common sources of that group.

- **DataOutPorts** — A DataOutPort is referenced by a connection property inside the Tdr/ DataInPorts wrapper when it is to be observed by the network. It is not referenced by the current IjtagNetwork when there is already a DataInPort in its fanout or the DataOutPort has the `icl_port` attribute “`tessent_use_in_dft_specification`” set to “false”. You can use the [set_ijtag_instance_options](#) command to force its inclusion or exclusion independent of whether it has a DataInPort in its fanout.

If the instance on which the DataOutPort exists has the `icl_port` attribute defined and set to true, the Tdr observing the DataOutPort will be inserted behind the Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the `keep_active_during_scan_test` attribute is “false”, then the Tdr observing the DataOutPort will be inserted below the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

Licensing

This command normally requires an IJTAG product license to run but there is one exception to this requirement. If the design contains one or more instances of an EDT/LBIST controller, the TestKompress/LogicBIST licenses are also allowed to run this command. In this case, the tool checks the availability of the license in the following order: IJTAG, TestKompress, LogicBIST. If none are currently available, the tool queues up for all three and uses the first license that becomes available.

You can refer to the [set_license_queue_timeout](#) and [get_license_queue_timeout](#) commands for more information about the license queuing feature.

Arguments

- **-existing_ijtag_host_scan_in *host_scan_in_design_pin_spec***

An optional switch and value pair that defines an existing host scan interface to which the entire IJtagNetwork should attach to inside the created [DftSpecification](#). The design pin must be on an instance having an associated ICL module and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. Specify this option during the first insertion pass. When unspecified, the interface to the ijtag network is determined using several criteria. When it is not the first insertion pass into the current design and either the Sib(sti) or the Sib(sri) is seen, it is automatically used as the connection point for the current insertion pass as illustrated in step 2 of the example below. If it is the first insertion pass, a TAP is inserted to host the root level Sib ring when design level is “chip”. The root level Sib ring is connected to ports when at the physical_block or sub_block level. The design level is specified as chip, physical_block or sub_block using the [set_design_level](#) command.

- **-existing_master_tap_scan_out *tap_client_scan_out_design_pin_spec***

An optional switch and value pair specifying that a current TAP client interface is to be used as master instead of creating a new one. It can only be used when the design level is “chip”. If other TAP or ijtag client interfaces, or if the dft requirements need additional ijtag nodes and this option is specified, you must also specify the **-existing_ijtag_host_scan_in** switch and value pair.

If the option is not used, the tool creates a TAP as the master TAP and additional TAP client interfaces already in the design are muxed—see “[Tap](#)” on page 3294 for more information. Each additional TAP client interface adds one scanMux and one DataOutPort in a new TDR (only a single TDR for many TAP client interface, 1 bit for each TAP.)

- **-existing_bscan_host_scan_in *bscan_host_scan_in_design_pin_spec***

An optional switch and value pair that defines an existing host bscan interface to which the boundary scan chain is to connect to. You typically do not need to specify this option. This switch value and value pair is mutually exclusive with the following switches:

- **-active_high_compliance_enables**
- **-active_low_compliance_enables**

If no TAP already exists in the design and the [set_design_level](#) command was used to specify level “chip”, a [Tap](#) wrapper will automatically be inserted in the [IJtagNetwork](#) wrapper to host both the ijtag network and the boundary scan chain. If a TAP controller was inserted in a previous insertion pass and you did not delete the HostBscan wrapper, it will automatically be seen and the boundary scan chain will automatically be hooked up to it. If you have your own third-party TAP controller and you described it with an ICL file as described in section [BoundaryScan](#) it too will be recognized by the command and the boundary scan chain will automatically be hooked up to it.

You specify this option if you have more than one TAP controller already present in the design that meets the requirements described in “[BoundaryScan](#)” on page 3384. You also

need to specify this option if the ICL description of your TAP does not call the ScanInterface associated with the host bscan interface named “bscan” exactly.

For example, the LogicVision TAP can be used to host the boundary scan chain, but if it was generated prior to version 2014.2, the ScanInterface for the boundary scan chain was not called “bscan”. In this case, you need to point to the “fromBscan” pin of the TAP using the -existing_bscan_host_scan_in option in order to have create_dft_specification command connect the boundary scan register to it.

- **-active_high_compliance_enables** *enable_port_name* ...

An optional switch and value pair that defines one or more ports that need to be set to 1 in order to access the TAP scan path that will be created with the resulting DftSpecification.

Use this option to create a Mentor TAP that shares the ports with a pre-existing connected TAP, which may or may not have an ICL description. This option can only be specified when design level is “chip” and cannot be used with -existing_bscan_host_scan_in, -existing_ijtag_host_scan_in, or -existing_master_tap_scan_out options.

For more information, see the [HostScanInterface](#) wrapper’s active_high_compliance_enables property and also “[Example 3](#)” on page 3247.

Note

 If the compliance enabling logic is already present in your design and you want to connect the TAP to it, follow the instructions described in [Example 2](#) of the [IjtagNetwork](#) wrapper section.

- **-active_low_compliance_enables** *enable_port_name* ...

An optional switch and value pair that defines one or more ports that need to be set to 0 in order to access the TAP scan path that will be created with the resulting DftSpecification.

Use this option to create a Mentor TAP that shares the ports with a pre-existing connected TAP that may or may not have an ICL description. This option can only be specified when design level is “chip” and cannot be used with -existing_bscan_host_scan_in, -existing_ijtag_host_scan_in, or -existing_master_tap_scan_out options.

Note

 If the compliance enabling logic is already present in your design and you want to connect the TAP to it, follow the instructions described in [Example 2](#) of the [IjtagNetwork](#) wrapper section.

- **-sri_sib_list** *sri_sib_list*

An optional switch and value pair that specifies a list of string ID that will be used to create Sib(id) wrappers in the created IjtagNetwork wrapper. The Scan Resource Instrument host Sib will be created if not already present. If it is already present, it will be referenced using the HostScanInterface wrapper and the requested Sibs will be inserted below it. This option is useful if you want to add your own instrument inside the network and this instrument needs to remain active during scan test. This option provides the host for your instrument in

the created DFTSpecification. This option is also useful to create a host for the OCC controllers. You run `create_dft_specification` with `-sri_sib_list occ` option, then you simply specify the `host_scan_interface` property in the OCC wrapper to Sib(occ) as shown in Example 2 below.

- `-sti_sib_list sti_sib_list`

An optional switch and value pair that specifies a list of string ID that will be used to create Sib(id) wrappers in the created IjtagNetwork wrapper. The Scan Tested Instrument host Sib will be created if not already present. If it is already present, it will be referenced using the HostScanInterface wrapper and the requested Sibs will be inserted below it. This option is useful if you want to add your own instrument inside the network and this instrument is to be tested as normal functional logic during scan test. This option provides the host for your instrument in the created DFTSpecification.

- `-replace`

An optional switch that suppresses the error that normally occurs if the `DftSpecification(design_name,id)` configuration wrapper already exists in memory. Instead, the existing one is deleted and replaced by the newly created one.

Examples

Example 1

The following example illustrates the created DftSpecification that is generated by the `create_dft_specification` command when running on different modules of a chip. The examples below are useful to understand how instruments are connected to the IjtagNetwork in a multi-insertion pass scenario.

Step 1

The first run is performed on a core called CoreA to create and hook up instruments to the network. These instruments are scan testable (`keep_active_during_scan_test` is “false”). It is the first dft insertion pass involving the IjtagNetwork in that module so the tool creates the Sib(sti) in this run.

```
set_context dft -rtl
#commands to load design appears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Sib(sti) is identified with the “`tessent_dft_function`” attribute set to “`scan_tested_instrument_host`”. All local scan interfaces are daisy chained below the Sib(sti_local).

```
DftSpecification(CoreA,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Sib(sti) {
                Attributes {
                    tesson_dft_function : scan_tested_instrument_host;
                }
                Sib(sti_local) {
                    DesignInstance(tdr_1) {
                        scan_interface : client;
                    }
                    DesignInstance(tdr_1) {
                        scan_interface : client;
                    }
                }
            }
        }
    }
}
```

Step 2

The second run is again performed on the core called CoreA at a later time to insert a Tdr used to control DataInBits on a scan control module that was user-inserted after the original insertion phase. The scan control module has an associated ICL file which documents its DataInPorts and with the ICL module attribute `keep_active_during_scan_test` set to “true”. Notice that because this is a second insertion pass into the same design module, the `design_id` is specified as `rtl2` in the `set_context` command to differentiate the insertion pass from the previous one. An error would have been generated had it not been specified because a conflict would have been noticed from the existence of instances of module names whose names start with “CoreA_rtl_tesson_”.

```
set_context dft -rtl -design_id rtl2
#commands to load design appears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Interface wrapper of the HostScanInterface wrapper is used to specify that the new Sib(sri) is to be inserted in series with the client interface of the Sib instance “CoreA_rtl_tesson_sib_sti_inst” which is the Sib(sti) created in the first insertion pass.

```
DftSpecification(CoreA,rtl2) {
    IjtagNetwork {
        HostScanInterface(sri) {
            Interface {
                design_instance : CoreA_rtl_tessent_sib_sti_inst;
                scan_interface : client;
            }
            Sib(sri) {
                Attributes {
                    tessent_dft_function : scan_resource_instrument_host;
                }
                Sib(sri_local) {
                    Tdr(sri_tdr1) {
                        DataOutPorts {
                            connection(0) : scan_control_inst/tm;
                            connection(1) : scan_control_inst/ac_mode;
                            connection(3:2) : scan_control_inst/config[1:0];
                        }
                    }
                }
            }
        }
    }
}
```

Step 3

The third run is performed on the chip called ChipA. It is used to provide access to the IJTAG scan interface of the 2 instances of CoreA and to again insert a Tdr to control the data_in ports of the scan control module situated at the top level. A Tap wrapper is created as the `-existing_ijtag_host_scan_in` option was not specified and the Tap from a previous insertion pass was not seen as this is the first insertion pass at the chip level.

```
set_context dft -rtl
#commands to load design appear here

set_current_design ChipA
set_design_level chip
set_attribute_value [get_ports TCK_p] -name function -value tck
set_attribute_value [get_ports TDI_p] -name function -value tdi
set_attribute_value [get_ports TDO_p] -name function -value tdo
set_attribute_value [get_ports TMS_p] -name function -value tms
set_attribute_value [get_ports TRST_p] -name function -value trst
set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. Notice how a Tap wrapper is being specified. It includes a HostBscan wrapper such that the TAP is ready for boundary scan-in a future insertion pass. One sib is inserted into the root level Sib ring to interface to the two instances of CoreA, and to provide access to the Tdr that is inserted to control the DataIn ports of the scan control module. Each child physical block is also wrapped by a local Sib to make sure other parts of the network are always accessible even if the physical block is defective, powered down, or in simulation, black-boxed.

create_dft_specification

```

DftSpecification(ChipA,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Interface {
                tck : TCK_p;
                trst : TRST_p;
                tms : TMS_p;
                tdi : TDI_p;
                tdo : TDO_p;
            }
        Tap(main) {
            HostBscan {
            }
            HostIjtag(1) {
                Sib(sri) {
                    Attributes {
                        tessent_dft_function : scan_resource_instrument_host;
                    }
                    Sib(pb1) {
                        DesignInstance(CoreA_I1) {
                            scan_interface : ijtag;
                        }
                    }
                    Sib(pb2) {
                        DesignInstance(CoreA_I2) {
                            scan_interface : ijtag;
                        }
                    }
                    Sib(sri_local) {
                        Tdr(sri_tdr1) {
                            DataOutPorts {
                                connection(0) : scan_control_inst/tm;
                                connection(1) : scan_control_inst/ac_mode;
                                connection(2) : scan_control_inst/config2;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Step 3 under [Example 1](#) could have been modified to connect to an existing TAP controller by simply using the `-existing_ijtag_host_scan_in` option as follows:

```
create_dft_specification -existing_ijtag_host_scan_in mytap/from_bist0
```

The created DftSpecification would then be as follows:

```
DftSpecification(ChipA,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Interface {
                DesignInstance (mytap)
                scan_interface : BistPort0;
            }
            Sib(sri) {
                Attributes {
                    tesseract_dft_function : scan_resource_instrument_host;
                }
                Sib(pb1) {
                    DesignInstance(CoreA_I1) {
                        scan_interface : ijtag;
                    }
                }
                Sib(pb2) {
                    DesignInstance(CoreA_I2) {
                        scan_interface : ijtag;
                    }
                }
                Sib(sri_local) {
                    Tdr(sri_tdr1) {
                        DataOutPorts {
                            connection(0) : scan_control_inst/tm;
                            connection(1) : scan_control_inst/ac_mode;
                            connection(2) : scan_control_inst/config2;
                        }
                    }
                }
            }
        }
    }
}
```

Example 2

This example shows how the `-sri_sib_list` option is used to create a host node for the OCC wrapper. The OCC wrapper makes reference back to the Sib(occ) using the `ijtag_host_interface` property.

```
set spec [create_dft_spec -sri_sib_list occ]
report_config_data $spec
```

create_dft_specification

```

DftSpecification(corea,gate) {
    use_rtl_cells : on;
    IjtagNetwork {
        ImplementationOptions {
            scan_path_retiming : flop;
        }
        HostScanInterface(ijtag) {
            Sib(sti) {
                Attributes {
                    tesson_dft_function : scan_tested_instrument_host;
                }
            Sib(sri) {
                Attributes {
                    tesson_dft_function : scan_resource_instrument_host;
                }
                Tdr(sri_ctrl) {
                    Attributes {
                        tesson_dft_function : scan_resource_instrument_dft_control;
                    }
                }
                Sib(occ) {
                }
            }
        }
    }
}

read_config_data -in_wrapper $spec -from_string {
    Occ {
        ijtag_host_interface : Sib(occ);
        Controller(c1) {
            clock_intercept_node : clk;
        }
    }
}

```

Example 3

The following example runs a script to push down each Tdr wrapper below its own Sib wrapper when there are more than one below the Sib(sti_local) and/or Sib(sri_local) wrapper.

```

proc push_tdr_below_sibs {spec} {
# Only works if the spec was created by create_dft_specification.
    set sib_locals [get_config_elements Sib(*_local) -below $spec -hier -silent]
    foreach_in_collection sib_local $sib_locals {
        set tdrs [get_config_element Tdr -below $sib_locals -silent]
        if {[sizeof_collection $tdrs] > 1} {
            foreach_in_collection tdr $tdrs {
                set id [get_config_value $tdr -id <0>]
                set sib [add_config_element Sib(sib_${id}) -in $sib_local]
                #Place New Sib in same parent instance as the TDR
                unset pin
                if {[get_config_value DataOutPorts/connection(0) -in $tdr -exist]} {
                    set pin [get_config_value DataOutPorts/connection(0) -in $tdr]
                } elseif {[get_config_value DataInPorts/connection(0) -in $tdr -exist]} {
                    set pin [get_config_value DataInPorts/connection(0) -in $tdr]
                }
                if {[info exists pin]} {
                    set parent_instance [get_attribute_value [get_pins [list $pin]] \
                        -name parent_instance]
                }
                if {$parent_instance ne ""} {
                    set_config_value parent_instance -in $sib $parent_instance
                    set_config_value parent_instance -in $tdr $parent_instance
                    move_config_element \
                        [get_config_value parent_instance -in $tdr -object]-first
                }
                move_config_element $tdr -in $sib
            }
        }
    }
    set spec [create_dft_specification]
    push_tdr_below_sibs $spec
}

```

You can see the Tdr wrapper before and after it was pushed below its own Sib Wrapper.

Before:

```

Tdr(sri_tdr4) {
    DataOutPorts {
        connection(0) : sub1_i2/sub2_i2/sub3_i1/di[0];
        connection(1) : sub1_i2/sub2_i2/sub3_i1/di[1];
        connection(2) : sub1_i2/sub2_i2/sub3_i1/di[2];
        connection(3) : sub1_i2/sub2_i2/sub3_i1/di[3];
        connection(4) : sub1_i2/sub2_i2/sub3_i1/di[4];
    }
}

```

After:

```
Sib(sib_sri_tdr4) {  
    parent_instance : sub1_i2/sub2_i2;  
    Tdr(sri_tdr4) {  
        parent_instance : sub1_i2/sub2_i2;  
        DataOutPorts {  
            connection(0) : sub1_i2/sub2_i2/sub3_i1/di[0];  
            connection(1) : sub1_i2/sub2_i2/sub3_i1/di[1];  
            connection(2) : sub1_i2/sub2_i2/sub3_i1/di[2];  
            connection(3) : sub1_i2/sub2_i2/sub3_i1/di[3];  
            connection(4) : sub1_i2/sub2_i2/sub3_i1/di[4];  
        }  
    }  
}
```

Related Topics

[apply_specification_defaults](#)
[process_dft_specification](#)
[report_config_syntax](#)
[set_design_level](#)

create_diagnosis_patterns

Context: patterns -scan_diagnosis

Mode: analysis

Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.

Usage

Logic Failure Diagnosis

```
create_diagnosis_patterns [-DIAGnosis_report {diagnosis_report1 diagnosis_report2 ...}]  
[-MAX_pattern_count number] [-TRAnsition]
```

Scan Chain Failure Diagnosis

```
create_diagnosis_patterns [-DIAGnosis_report {diagnosis_report}] [-CHAIN chain_name  
[-CELL_range lowerbound upperbound] [-FAULT_type {stuck_at_0 | stuck_at_1 |  
slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast}]]
```

Description

Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.

For usage cases using this command, refer to “[Iterative Logic Diagnosis Examples](#)” in the *Tessent Diagnosis User’s Manual*. You can use iterative diagnosis with the layout-aware diagnosis flow as long as you use the layout in the same iterative diagnosis session—see “[Layout-Aware Diagnosis](#)” in the *Tessent Diagnosis User’s Manual*.

Note

 After issuing this command, you should write out the test patterns to an external file using the write_patterns command.

Logic Failure Diagnosis: The tool loads the diagnosis result from a prior run or a diagnosis report file, create a fault list, and generates the patterns targeting suspect signals in the diagnosis report.

Scan Chain Failure Diagnosis: The tool loads either of the following inputs you specify:

- A diagnosis report file containing the diagnosis results of scan chain failure.
- The chain names, scan cell range, and the fault type.

For scan chain failure diagnosis, you can also provide a chain name (using the –chain switch) and a diagnosis report containing multiple faulty scan chains. In this case, the diagnostic test generation only creates patterns for the scan chain specified by the –Chain switch.

Arguments

Logic Failure Diagnosis Arguments

- **-DIAGnosis_report** {*diagnosis_report1 diagnosis_report2 ...*}

An optional switch and one or more strings that specifies the diagnosis report file(s) to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. You can also supply more than one diagnosis report for suspects from multiple diagnosis report files are targeted in one diagnosis test generation run. The diagnosis report must be in the default ASCII format.

- **-MAX_pattern_count** *number*

An optional switch and integer pair that specifies the maximum pattern count the tool generates. The default is no limit. Use when ATPG run time is a concern. The limit defined by **-MAX_pattern_count** is applied per chain if used for chain pattern generation.

- **-TRAnsition**

An optional switch that specifies creating transition test patterns. By default, the transition patterns generated are launch off capture (equivalent to the old `no_shiftlaunch` option).

Scan Chain Failure Diagnosis Arguments

- **-DIAGnosis_report** {*diagnosis_report*}

An optional switch and string pair that specifies the diagnosis report file to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. Note that only a single diagnosis report is allowed as input for chain failures.

- **-CHAIN** *chain_name*

An optional argument and string pair that specifies a single chain.

- **-CELL_range** *lowerbound upperbound*

An optional argument and pair of integers for the **-chain** option that specifies the beginning and ending of a range of cells.

- **-FAULT_type** {*stuck_at_0 | stuck_at_1 | slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast*}

An optional argument and literal pair for the **-chain** option that specifies the fault type.

create_feature_statistics

Context: patterns -scan_diagnosis

Mode: analysis

Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.

Usage

```
create_feature_statistics [-temp_dir dir_name] [-add_processors {hostname[:{cpus |  
MAXcpu}]}...] [-force]
```

Description

Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.

The tool analyzes the design for all possible diagnosis suspects for certain features in the layout. It determines which bridge pairs, net segments, and cells are tested, and extracts the relevant physical information such as the critical area, via count, and cell instance count for each possible suspect.

You must create the RCD constants statistics for every flat model and pattern combination. To do this, you can process the combinations in parallel in different tool sessions.

RCD analysis in Tessent YieldInsight determines the probability sums for the features across the diagnoses and uses the RCD constants to normalize their weights to account for diagnosis noise.

To use this command, you must have previously loaded your patterns and opened a LDB. Otherwise, the tool issues an error message. The feature statistics are stored in the LDB for later use by Tessent YieldInsight RCD analysis. They are not used during the diagnosis process.

For more information, see “[Diagnosis for Root Cause Deconvolution Analysis](#)” in the *Tessent Diagnosis User’s Guide*.

Arguments

- **-temp_dir *dir_name***
An optional switch and string pair that specifies a temporary directory to use for command execution.
- **-add_processors {*hostname*[:{*cpus*| MAXcpu}]}...**
An optional switch and string that specifies to run multiple processors in parallel on multiple machines to reduce the runtime for calculating the RCD constants. The tool calls the [add_processors](#) command and uses a multithreading process as described for the [add_processors](#) command.

Although the -add_processors switch supports the same options as the add_processors command, for the best performance, specify the localhost with 8 processors.

- -force

An optional switch that specifies to delete from the LDB the temp file or .rcddb file associated with the current flat model before attempting to generate feature statistics. This allows you to re-run RCD creation in the event of a premature termination, which can result in temporary locked files. The tool deletes the temp files, issues a warning and continues processing.

Examples

Example 1

The following example loads patterns and opens the layout before calculating the feature statistics.

```
read_patterns external my_test_patterns.wgl
//  Reading WGL input file "my_test_patterns.wgl"

open_layout my_layout.db
//  DieArea: minx=-2107.000000, miny=-2002.000000, maxx=2107.000000,
//    maxy=2002.000000 in microns
//  Note: Checking layout consistency...
//  Note: Layout Database opened successfully.

create_feature_statistics
// Determining feature detection information
// 10% Done.
// 22% Done.
.....
// 100% Done.
// Determining Open Feature Statistics
// 11% Done.
// 21% Done.
.....
// 100% Done.
// Determining Bridge Feature Statistics
// 13% Done.
.....
// Feature Statistics Calculation Complete. Information is written into
// the layout database.
```

Example 2

The following example shows the message that the tool returns when you issue the create_feature_statistics when the feature statistics were previously calculated and stored in the LDB.

```
create_feature_statistics
```

```
// Note: Feature statistics for this flat model and pattern set have  
// already been calculated and stored in the layout database. There is no  
// need to perform this operation again.
```

create_flat_model

Context: all contexts

Mode: setup

Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the DFTVisualizer Flat Schematic window.

Usage

`create_flat_model`

Description

Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the DFTVisualizer Flat Schematic window.

The tool automatically flattens the design hierarchy down to the logically equivalent design when you exit Setup mode. However, there may be times that you would like to access the flat model without having to exit Setup mode. For example, you may want to add ATPG constraints and functions before you exit Setup mode.

If you exit Setup mode and then add ATPG constraints and functions, the design rule checker does not have access to those ATPG constraints during the rule checking. If you issue the `create_flat_model` command in Setup mode and then add those ATPG constraints, the design rule checker has access to them during the rule checking.

Note

 If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (issuing the `create_patterns` command) and saving final patterns. For more information, refer to “[Preparing the Design Netlist](#)” in the *Tessent Diagnosis User's Manual*.

Arguments

None

Examples

The following example shows flattening the design to the simulation primitives before adding constraints that the rule checker then uses when you run the design rule checker. The rule checker runs when you first attempt to exit Setup mode

```
create_flat_model
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
set_system_mode analysis
```

The following example sets the system mode to setup, creates a flat model, and opens DFTVisualizer:

```
set_system_mode setup
create_flat_model
open_visualizer
```

Related Topics

[set_system_mode](#)
[delete_atpg_functions](#)
[write_flat_model](#)
[add_atpg_constraints](#)

create_icl_verification_patterns

Context: patterns -ijtag

Mode: analysis

Prerequisites: To use this command, you must have previously opened a pattern set with the open_pattern_set command.

Specifies the creation of a structural pattern to verify the proper operation of the IJTAG network.

Usage

```
create_icl_verification_patterns [-scan_test {on | off}]  
      [-data_pin_test {on | off | end_points_only}]  
      [-modules modules] [-exclude_modules exclude_modules]  
      [-instances instances] [-exclude_instances exclude_instances]  
      [-exclude_scan_registers exclude_scan_registers]  
      [-exclude_data_pins exclude_data_pins]
```

Description

Specifies the creation of a structural pattern set to verify the proper operation of the IJTAG network. In simulation, the pattern set is useful for identifying mismatches between the ICL and Verilog model views. In manufacturing test, the pattern set covers manufacturing defects that can exist in the IJTAG network circuitry. You do not need to use this command if you are using the more automated [create_patterns_specification/process_patterns_specification](#) flow. For more information about this flow, see the [ICLNetworkVerify](#) wrapper.

When this command is run with the -scan_test option set to on, the tool creates verification patterns by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way; this exercises all scan configurations. On the first scan load, the predictably captured values of the ScanRegisters are compared, and the scan registers are loaded with all 0s with a single 1 closest to ScanIn. The ScanRegister is then parked in pause state and unloaded to make sure the same data comes out; this ensures that the ScanRegisters are actually the length documented in the ICL model.

When the -data_pin_test option is set to on, any internal DataOutPorts that do not have a source described in ICL are forced, and those values are observed by ScanRegisters unloading. The internal DataInPorts that do not have a destination described in ICL are controlled using ScanRegisters loading, and the resulting value is directly observed on the internal DataInPorts. Note that this feature is only usable in simulation where it is useful for finding mismatches between the circuits and their ICL descriptions.

Arguments

- **-scan_test {on|off}**

An optional switch that specifies whether the patterns are to verify the integrity of the scan paths. When the option is set to on, the tool creates the patterns by tracing backward from all

ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictably captured values of the ScanRegisters are compared and the scan registers are loaded with all 0s with a single 1 closest to ScanIn. The ScanRegister is then parked in pause state and unloaded to make sure the same data comes out; this ensures that the ScanRegisters are actually the length documented in the ICL model. The opposite pair of patterns are also applied as shown here:

```
load 100000...000 expect captures values described in ICL
load 000000...000 expect 100000...000
load 011111...111 expect captures values described in ICL
load 111111...111 expect 011111...111
```

- **-data_pin_test {on | off | end_points_only}**

An optional switch that specifies whether the patterns are to verify the values observed by Internal DataInPorts and controlled by internal DataOutPorts.

end_points_only—With this value, the internal DataOutPorts that do not have a source described in ICL are forced, and the forced values observed using the IJTAG network. The internal DataInPorts that do not have a destination described in ICL are controlled using the IJTAG network, and the resulting value is directly observed on the internal DataInPorts.

On—With this value, not only are the DataInPorts with no destination and the DataOutPorts with no source observed and controlled, but all DataInPorts having an observation path through the IJTAG network are forced in the test bench and observed by the IJTAG network to confirm that the ICL and HDL descriptions are consistent. Also, all DataInPorts having some controllability from the network are sensitized to a one and zero value and that value is directly observed on the DataInPort in the test bench.

Off—With this value, no force and compare are done on the DataInPorts and DataOutPorts.

Note that this feature is only usable in simulation where it is useful for finding mismatches between the HDL and associated ICL descriptions. For this feature to be usable, the ICL should have been created using `extract_icl` such that each ICL instance has the `tesson_design_instance` attribute defined to reflect the original design instance path associated to the ICL instance.

Also, note that ICL Extraction sets the `tesson_design_instance` attribute to the pre-synthesis instance names whenever those names are available. During the creation of the data pin verification patterns, the actual design instance names are ignored. The verification pattern generator only uses the `tesson_design_instance` attribute to obtain the references to the design pins. Consequently, the rtl design files provided to ICL extraction must be used during the simulation of the verification patterns, at least in case that the synthesis modifies the design hierarchy (for example, because of “generate” loops). In order to obtain data pin verification patterns which can be simulated using the synthesized netlist with instance names that have been modified by synthesis, you either have to modify the `tesson_design_instance` attributes accordingly or run ICL Extraction based on the synthesized netlist from the beginning (without providing rtl files to the tool).

- **-modules *modules***

An optional switch that specifies the ICL modules the test should focus on. The modules parameter value contains glob patterns matching ICL module names instantiated below the current ICL design. For more information on glob pattern matching, refer to “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164.

- **-exclude_modules *exclude_modules***

An optional switch that specifies the ICL modules the test should exclude. The exclude_modules parameter value contains glob patterns matching ICL module names instantiated below the current ICL design. For more information on glob pattern matching, refer to “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164.

ICL modules that have the tessent_ignore_during_icl_verification attribute set to on are automatically excluded. You can use this attribute on ICL modules that are not truly IJTAG compliant and cannot pass the structural test performed in this test. One such example is the Tessent MemoryBIST controller. Because of its asynchronous interface, its BIST_CLOCK must be toggling at least four times faster than TCK to enable a proper shift operation. Because this dependency on the ClockPort is not documented in ICL, the ICLNetworkVerify does not know to enable the source of the clock port. These modules are tested as part of the memory BIST operations. They are also tested during scan test because the memory BIST controllers are scan testable.

- **-instances *instances***

An optional switch that specifies the only ICL instances the test should be performed on. The instances parameter value contains glob patterns matching ICL instances found below the current ICL design. For more information on glob pattern matching, refer to “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164.

- **-exclude_instances *exclude_instances***

An optional switch that specifies the set of ICL instances to exclude from the test. The exclude_instances parameter value contains glob patterns matching ICL instances found below the current ICL design. ICL instances that have the tessent_ignore_during_icl_verification attribute set to on are automatically excluded.

- **-exclude_scan_registers *exclude_scan_registers***

An optional switch that specifies the set of ICL scan registers to exclude from the test. The exclude_scan_registers parameter value contains glob patterns matching ICL instances found below the current ICL design. ICL Scan Registers that have the tessent_ignore_during_icl_verification attribute set to on are automatically excluded. This switch is used in the LogicBIST ICL module to exclude the single chain diagnostic registers because they are too long to be simulated serially. This switch is only used when -scan_test is set to on.

- **-exclude_data_pins *exclude_data_pins***

An optional switch that specifies the set of ICL pins to exclude from the data pin test. The exclude_data_pins parameter value contains glob patterns matching ICL pins found below

the current ICL design. ICL DataInPorts and DataOutPorts that have the `tessent_ignore_during_icl_verification` attribute set to on are automatically excluded. This switch is only used when `-data_pin_test` is set to on.

Examples

The following example loads an ICL file, generates ICL verification patterns, and writes out a simulation pattern file. Because this process is automated as part of the [create_patterns_specification](#) and [process_patterns_specification](#) flow which also provides simulation automation using the [run_testbench_simulations](#) command, it typically does not need to be performed as shown here.

```
set_context patterns -ijtag
read_icl my_instrument.icl
set_current_design
set_system_mode analysis
open_pattern_set icl_verify
create_icl_verification_patterns -data_pin_test on
close_pattern_set
write_pattern icl_verify.v -verilog
```

Related Topics

[ICLNetworkVerify](#)
[process_patterns_specification](#)
[create_patterns_specification](#)
[run_testbench_simulations](#)

create_initialization_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Creates RAM initialization patterns and places them in the internal pattern set.

Usage

create_initialization_patterns *RAM_instance_name* | *RAM_gate_id#*

Description

Creates RAM initialization patterns and places them in the internal pattern set.

The `create_initialization_patterns` command creates RAM initialization patterns that write values into the specified RAM. The tool places these patterns into the internal pattern set from which you may save them into a file.

You can identify the RAM by its instance name or gate ID number. An error condition occurs if there is not a single RAM gate inside the instance or if the specified gate is not a RAM. An error condition also occurs if the RAM does not have an initialization file, or if the RAM did not successfully pass stability checking (Design Rules A1 and A6).

The tool creates the initialization patterns by doing an independent test generation for each valid address which has a non-X state on at least one data line. If the test generation aborts, the command terminates with an error message. The patterns contain a measure PO statement, but all values are X. There may also be an unload statement, but all values are X. If the patterns are re-simulated (as in pattern compression), the tool deletes patterns that do not detect faults. For those the tool does not delete, simulation values replace the Xs in the measure PO statement.

Arguments

- ***RAM_instance_name***

A string that specifies the instance name of the RAM for which you want to `create_initialization_patterns`.

- ***RAM_gate_id#***

An integer that specifies the gate identification number of the RAM for which you want to `create_initialization_patterns`.

Examples

The following example creates RAM initialization patterns for p1.ram/u1, places the patterns into the internal pattern set during the ATPG run, and saves the patterns to a pattern file with the name *patfile*:

```
add_write_controls 0 w1
set_system_mode analysis
add_faults -all
create_initialization_patterns p1.ram/u1
create_patterns
write_patterns patfile
```

Related Topics

[read_modelfile](#)

[write_modelfile](#)

create_instance

Context: all contexts

Mode: insertion

Instantiates an instance of a module mod_spec inside a design module in the current design.

Usage

```
create_instance hierInstPath -of_module mod_spec
    [-below_instance instance_object]
    [-allow_instance_uniquification [-instance_uniquification_suffix string]]
    [-parameter_values parameter_list]
    [-silent]
```

Description

Instantiates an instance of a module mod_spec inside a design module in the current design.

If the command is successful, this command returns a collection containing the newly created instance.

The pathname to the location must exist in the design up to the leaf level. If the leaf level already exists and the -allow_instance_uniquification switch is specified, Tesson Shell adds a uniquification suffix to the new instance to create a unique instance name. The uniquification suffix defaults to `_#` where # is replaced by the first available integer starting from 1. You can specify a different uniquification suffix using the -instance_uniquification_suffix option in the create_instance command or in the [set_insertion_options](#) command.

In addition, ports of the module that have the [function](#) attribute set to “tie0” or “tie1” are automatically tied on the created instance.

Arguments

- ***hierInstPath***

A required string that specifies the hierarchical path in which the instance is to be created concatenated to the leaf instance name of the new instance. The specified hierarchical path in which the instance is to be created must exist in the design. If an instance of the module to be instantiated already exists in that module, by default the tool unifies the new name by adding a “`_#`” suffix to the instance name. You can use the instance_uniquification_suffix switch to specify a different suffix.

- ***-of_module mod_spec***

A required switch and value pair that specifies the module to be instantiated. mod_spec can be the name of a module or a module object returned by the [get_modules](#) command.

Note

Currently, any instantiation of parameterized modules can only be done with default values.

- **-below_instance *instance_object***

An optional switch and value pair that specifies to create the instance below the specified *instance_object*. The instance specified with the ***hierInstPath*** value is created below the specified instance using the **-below_instance** switch. Using this switch enables you to create instances below a given instance without having to extract the name of that instance and use it as the beginning part of the ***hierInstPath*** value with a “/” to separate the two parts.

- **-allow_instance_uniquification**

An optional switch that specifies to automatically uniquify the name of the new instance being instantiated if an instance of the same name already exists in the current design. This feature is very useful when inserting gates to intercept signals. You can always use the same leaf instance name and let the tool resolve conflicts automatically. This use model is illustrated in Examples 2, 3, 4 and 5 in the [move_connections](#) command description.

- **-instance_uniquification_suffix *string***

An optional switch that specifies a string to add to the new instance name when an instance of the same name already exists and you specify the **-allow_instance_uniquification** switch. If you don't specify this switch, the tool automatically uniquifies the object using the default string “_#” where # begins at 1 and increments by one until a unique name has been found. The string must always end with a # to denote the uniquification integer. You can also specify a different uniquification suffix using the [set_insertion_options](#) command.

- **-parameter_values *parameter_list***

An optional switch that specifies a Tcl list of parameter values when you use the command to create an instance of a parameterized module. The instance uses the supplied list of parameter values. For an example of how to use this switch, refer to “[Example 4](#)” on page 506.

This switch is available only in -rtl mode (“`set_context dft -rtl`”). For more information, refer to the [set_context](#) command.

- **-silent**

An optional argument that specifies to not generate an error message under the following condition. When the error is suppressed, no instance is created and the command returns a null collection. When no errors are suppressed, the command returns a collection containing the newly created instance.

- The hierarchical path above the leaf instance name does not exist in the design.
- The specified mod_spec is not the name of an existing module.
- The collection supplied to the **-of_module** option does not contain a single module object.

Examples

Example 1

The following example creates an instance of module ModA called ModA_1 inside instance Block1_I1/Block2_I1. The return value is an instance object by the name of Block1_I1/Block2_I1/ModA_1.

```
create_instance Block1_I1/Block2_I1/ModA_1 -of_module ModA
{Block1_I1/Block2_I1/ModA_1}
```

Example 2

The following example creates two instances of module MUX called MUX inside instance Block1_I1. The tool unifies the leaf instance name when a name conflict occurs on the second invocation of the command.

```
create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification
{Block1_I1/MUX}

create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification
{Block1_I1/MUX_1 }
```

Example 3

This example creates two instances of module MUX called MUX inside instance Block1_I1. The tool unifies the leaf instance name when a name conflict occurs. The get_attribute_value_list command returns the value of the leaf_name attribute on the new instance name.

```
get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name
{MUX}

get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name
{MUX_1}
```

Example 4

This example creates an instance of a parameterized module using a list of parameter values. This example is based on the following Verilog module:

```
module ff(clk,d,q);
    parameter SIZE = 1;
    parameter OTHER = 0;
    input          clk;
    input  [(SIZE-1):0] d;
    output [(SIZE-1):0] q;
    reg   [(SIZE-1):0] q;
    ...
}
```

Then the following command

```
create_instance ff_i3 -of_module ff -parameter_values { SIZE 10 }
```

creates the following Verilog code:

```
ff #(SIZE(10), OTHER(0)) ff_i3(.clk(), .d(), .q());
```

And this command

```
create_instance ff_i4 -of_module ff -parameter_values [list SIZE 10 OTHER 10]
```

creates the following Verilog code:

```
ff #(SIZE(10), OTHER(10)) ff_i4(.clk(), .d(), .q());
```

Related Topics

[copy_module](#)

[delete_instances](#)

[get_common_parent_instance](#)

[get_nets](#)

[get_icl_pins](#)

[get_icl_ports](#)

[rename_instance](#)

[replace_instances](#)

[uniquify_instances](#)

create_layout

Context: patterns -scan_diagnosis

Mode: analysis

Generates a diagnosis tool-compatible Layout Database directory (LDB) from LEF/DEF input files.

Usage

```
create_layout layout_database_name [ -lef lef_file_name ... | -leflist leflist ]
  [-def def_file_name ... | -deflist deflist]
  {[-extra_layout_hierarchy extra_levels_of_hierarchy_in_layout] | [-extra_design_hierarchy
extra_levels_of_hierarchy_in_design]}
  [-threshold percentage_match] [-topology_threshold percentage_match]
  [-temp_dir directory_name] [-compression {on | off}]
  [-keep_uncompressed_database] [-core core_name] [-verify_only]
  [-threads num_threads [-min_threads min_threads]]
  [-compact {on | off}] [-bridge_proximity_factor bridge_proximity_factor]
  [-do_tolerant_match {on | off}] [-topology_error_count integer]
  [-gray_box_detection] [-replace] [-continue]
```

Description

Generates a diagnosis tool-compatible LDB from LEF/DEF input files.

See “[Layout-Aware Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

Before generating the LDB, the tool performs layout verification of the LEF/DEF files and reports any mismatches to the transcript—see “[Layout Verification and Layout Database Creation Process](#)” in the *Tessent Diagnosis User’s Manual*.

After the layout-verification stage, you can interrupt the LDB creation by issuing a **Ctrl-C** in the invocation shell. You can subsequently use the [report_layout_rules](#) command to obtain details on the layout rule violations and use this information to correct errors in LEF/DEF files, providing the correct LEF/DEF files and/or add any missing LEF/DEF files to the list provided to the `create_layout` command. Following this, you then use the `create_layout` command to determine if there are any additional issues. Use this method in iterative fashion to correct any problems with the LEF/DEF before creating the LDB.

After generating the LDB, the tool performs layout pre-extraction. The tool extracts bridge and open defect information and stores this information in the LDB for use with RCD constant creation (with the [create_feature_statistics](#) command) and dynamic partitioning-based diagnosis. In addition, layout pre-extraction improves layout-aware diagnosis performance.

You can improve runtime performance of the layout pre-extraction step by specifying the `-threads` and `-min_threads` options. These options allow you to check out additional licenses so that Tesseract Diagnosis can perform some layout pre-extraction tasks in parallel.

Tip

i For maximum efficiency, run the `create_layout` command on a disk that is local to the machine hosting the process. Additionally, since by default this command reads the entire physical layout information into memory, you should run it on a machine with sufficient physical memory.

Arguments

- *layout_database_name*

A required string that specifies the name of the LDB that diagnosis creates from the LEF/DEF input.

- `-lef lef_file_name ...`

An optional switch and string that specifies space-separated name(s) of the LEF file(s).

- `-leflist leflist`

An optional switch and string pair that specifies a TCL list of the LEF file(s).

- `-def def_file_name ...`

An optional switch and string that specifies space-separated name(s) of the DEF file(s).

- `-deflist deflist`

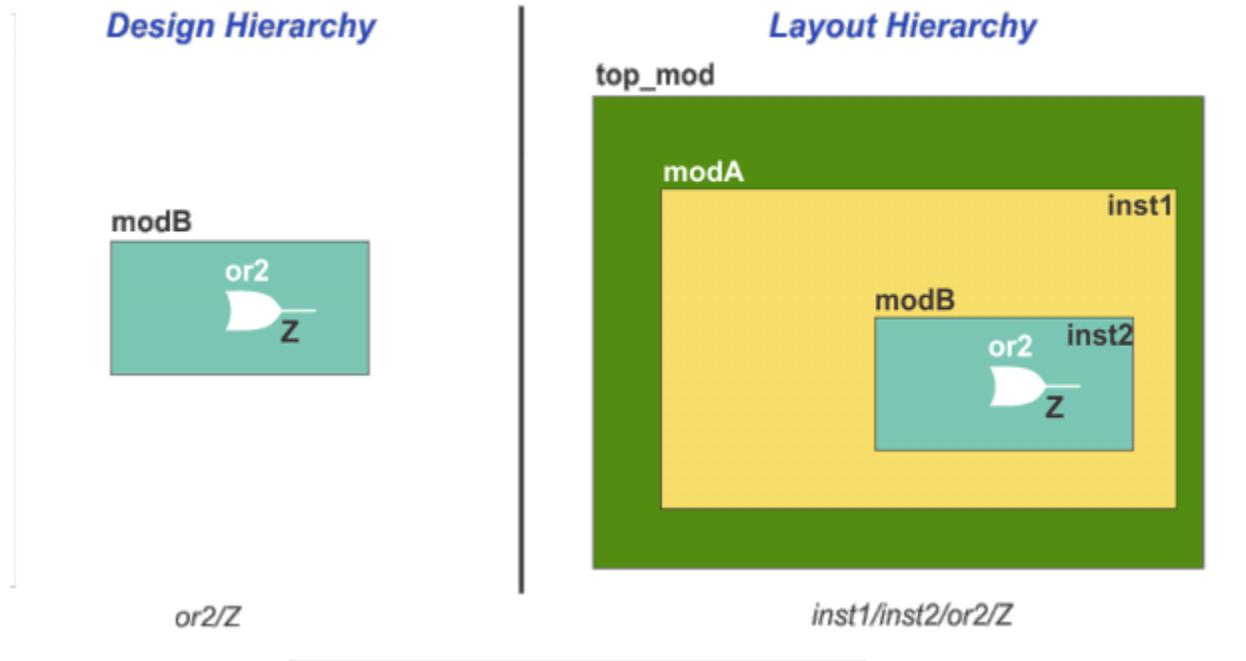
An optional switch and string pair that specifies a TCL list of the DEF file(s).

- `-extra_layout_hierarchy extra_levels_of_hierarchy_in_layout`

An optional switch and string pair that removes extra levels of layout hierarchy such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

You use this switch if your design is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. [Figure 3-24](#) illustrates this usage.

Figure 3-24. Extra Layout Hierarchy Example



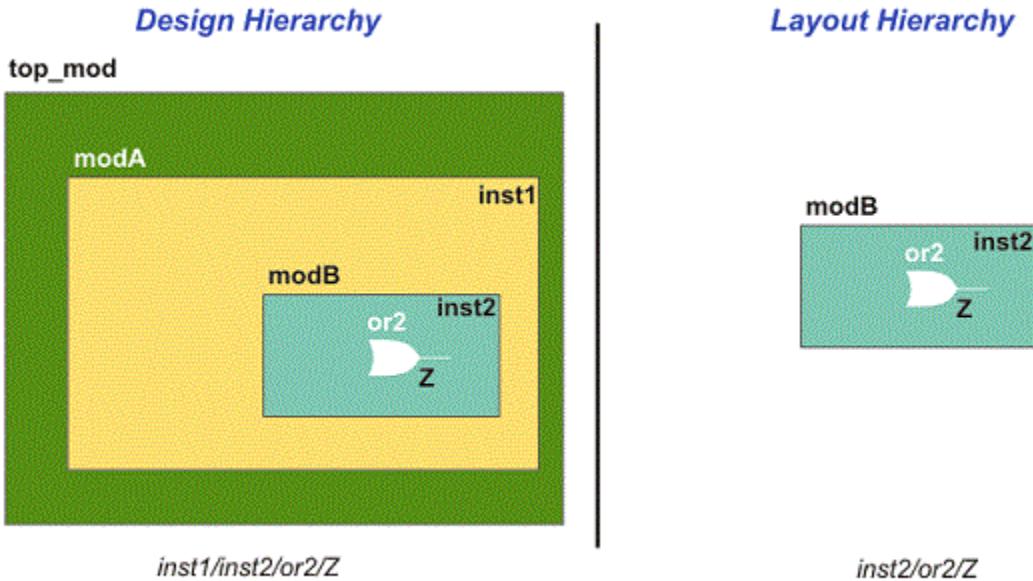
In the case of [Figure 3-24](#), the following command removes the hierarchy from the LEF/DEF net, pin names, and so forth before matching with the design name:

```
create_layout layout_database_name -lef lef_file -def def_file \
-exTRA_lAyout_hIerarChy inst1/inst2
```

- **-extra_design_hierarchy extra_levels_of_hierarchy_in_design**

An optional switch and string pair removes extra levels of design hierarchy such that the layout hierarchy and design hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

You use this switch if your layout is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. [Figure 3-25](#) illustrates this usage.

Figure 3-25. Extra Design Hierarchy Example

In the case of [Figure 3-25](#), the following command removes hierarchy from a design pin, net name, and so forth before matching with layout:

```
create_layout layout_database_name -lef lef_file -def def_file \
-exTRA_design_hierarChy inst1
```

Note

The -EXTRA_DESign_hierarChy and -EXTRA_LAYout_hierarChy switches are mutually exclusive, and the tool issues an error if you specify both.

- -threshold *percentage_match*

An optional switch and integer pair that specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent creation of the LDB. The default threshold value is 85 percent. The following command sets the required layout match threshold to 92%:

```
create_layout design.layout -lef lef_input.lef -def def_input.def \
-threshold 92
```

- -topology_threshold *percentage_match*

An optional switch and integer pair that specifies the minimum percentage net tracing threshold required to successfully complete topology extraction during the LDB creation process. The default threshold value is 90 percent, meaning topology extraction is considered successful if the tool can trace 90% of the nets. If the success rate falls below the

default or user-specified *percentage_match* rate, the tool issues an error and stops LDB creation.

- **-temp_dir *directory_name***

In some cases, Tesson Diagnosis must update the LDB (for example, if you use the **-verify** switch). Due to the fact that the compressed LDB is a read-only database, Tesson Diagnosis automatically uncompresses, updates, and re-compresses the LDB. By default, the temporary uncompressed LABD is stored into the same location where the compressed LDB is located; if you have limited disk space in this location, then use the **-TEmp_dir** switch and string pair to specify an alternative location.

See “[Layout Database Compression and Decompression](#)” in the *Tesson Diagnosis User's Manual* for complete information.

- **-compression {on | off}**

An optional switch and literal that controls whether the tool compresses the LDB during database creation. The default is OFF. Specify ON to create a compressed LDB.

See “[Layout Database Compression and Decompression](#)” in the *Tesson Diagnosis User's Manual* for complete information.

- **-keep_uncompressed_database**

An optional switch that specifies that the tool retain a copy of the uncompressed LDB as well as a compressed version of the LDB.

- **-core *core_name***

An optional switch that specifies to create an instance-aware core-level LDB for the specified core design. This option only pertains to and is required for the hierarchical layout-aware diagnosis flow.

See “[Diagnosis for Hierarchical Design](#)” in the *Tesson Diagnosis User's Manual* for complete information.

- **-verify_only**

An optional switch that enables LEF/DEF design verification and stops LDB creation after verification steps have completed.

- **-threads *num_threads***

An optional switch and integer pair that specifies the number of CPUs to use so that the tool can run some `create_layout` tasks in parallel.

num_threads is an integer such that:

num_threads > 1: Use the specified number of threads.

num_threads = 0: Use the number of CPUs of the host for the number of threads.

num_threads < 1: Use the number of CPUs less the specified *num_threads* value.

If the requested number of threads is greater than the host's CPUs, the tool automatically adjusts the thread count down to the number of CPUs.

The tool requires one yieldascandiag license per thread. For example, if you specify “-threads 8”, the tool checks out 7 yieldascandiag licenses in addition to the 1 yieldascandiag license that was checked out when you entered scan_diagnosis context. The tool releases the additional licenses when the `create_layout` command completes processing.

If less than the requested licenses are available, license queuing occurs based on the setting from the `set_license_queue_timeout` command. If the tool cannot acquire the number of licenses specified by the user, the number of threads is adjusted down to the number of acquired licenses.

- **`-min_threads min_threads`**

An optional switch and integer pair that is used with the `-threads` option and specifies the minimum number of threads that must be available. The tool aborts the `create_layout` command if the number of acquired licenses does not meet the `-min_threads` requirement.

num_threads is an integer such that:

num_threads > 1: Use the specified number of threads.

num_threads = 0: Use the number of CPUs of the host for the number of threads.

num_threads < 1: Use the number of CPUs less the specified *num_threads* value.

- **`-compact {on | off}`**

An optional switch that specifies whether to produce a compact LDB, which is an optimized database that is faster to create and has a smaller size on disk. Compact LDBs have the following limitations:

- In the Tessent YieldInsight layout viewer, you will not see background polygons. (Landmark and suspect polygon display remains the same.)
- The tool does not perform DFM rule verification during the `import_dfm` process. (The rest of the DFM flow and reporting remains the same.)

You can convert existing compact LDBs by specifying “`create_layout existing_LDB -compact off`.”

- **`-bridge_proximity_factor bridge_proximity_factor`**

An optional switch that controls the maximum distance allowed between nets for them to be considered possible bridge defects. Specify a scale factor from 1 to 100. The default is 10. The tool calculates the maximum distance by multiplying the specified scale factor by the minimum net width for the layer.

This option is useful for eliminating bridge suspects that have minimal significance for failure analysis thus improving resolution metrics.

- **`-do_tolerant_match {on | off}`**

An optional switch and literal that specifies whether to allow escape-tolerant name matching during layout verification. By default, the tool tolerates differences in escapes in instance path names and net path names. This switch only applies to name matching during

verification and has no impact to names stored in the LDB. The names stored in the LDB are the original LEF/DEF names.

- **-topology_error_count *integer***

An optional switch and integer that specifies the number of net errors per issue class to display in the net topology extraction transcript. The default is 5.

- **-gray_box_detection**

An optional switch that specifies to populate the LDB with graybox layout information for the purposes of performing graybox-aware top-level pattern diagnosis. For more information, see “[Top-Level Layout-Aware Diagnosis](#)” in the *Tessent Diagnosis User’s Manual*.

- **-replace**

An optional switch that overwrites existing layout pre-extraction information stored in the LDB with newly generated layout pre-extraction files.

- **-continue**

An optional switch that specifies to continue generating the LDB, including the layout open and bridge defect information. Use this switch when:

- Processing halts for any reason. The tool automatically re-starts processing where it left off.
- You have a pre-2014.4 LDB and you would like to add layout open and bridge defect information to it.

Examples

Example 1

In the following example, the diagnosis tool creates a LDB (*design.layout*) directory that includes pre-extracted bridge and open information from multiple LEF files and DEF files:

```
create_layout design.layout -lef lef_input01.lef lef_input02.lef -def def_01.def def_02.def
```

Example 2

In the following example, Tessent Diagnosis creates a LDB (*design.layout*) directory that includes pre-extracted bridge and open information from a LEF file (*lef_input.lef*) and four DEF files (*a.def*, *b.def*, *c.def*, *d.def*). It skips re-generating valid pre-extraction information.

```
create_layout design.layout -lef lef_input.lef -def a.def b.def c.def d.def -continue
```

Example 3

In the following example, the tool pre-extracts layout bridge and open defect information and stores it in an existing LDB. It recreates all the layout pre-extraction information.

```
create_layout design.layout -replace
```

Example 4

In the following example, the set_license_queue_timeout is set for a one minute timeout and you have specified to check out four licenses (for four threads) with the create_layout command.

```
set_license_queue_timeout 1
create_layout results/layoutDB -def ya_demo6.def.gz -lef ya_demo5.lef.gz -threads 4
// Note: multithreading license queue type: timeout wait
// Note: Requested threads (4) requires check out of 3 additional
// licenses.
// Note: Acquired 2 of 3 requested licenses.
// Note: Queueing 1 minute for 1 license ( CTRL-C to run with available
// licenses ).
// Waited 1 of 1 minute for 'yieldascandiag'.
// Warning: Could not acquire 3 licenses. Proceeding with reduced thread
// count of 2.
```

Example 5

In the following example, the set_license_queue_timeout is set for no waiting and you have specified that you require four licenses. However, there are only three available licenses.

```
set_license_queue_timeout no_queue
create_layout results/layoutDB5 -def ya_demo6.def.gz -lef ya_demo5.lef.gz -threads \
4 -min_threads 4
// Note: multithreading license queue type: no_queue
// Note: Requested threads (4) requires check out of 3 additional
// licenses.
// Note: Acquired 2 of 3 requested licenses.
// Error: Insufficient licenses acquired (2) and -min_threads (4)
// specified.
// Note: Releasing 2 licenses.
// Error: Layout feature preparation failed.
```

Related Topics

[close_layout](#)
[compress_layout](#)
[open_layout](#)
[report_layout_rules](#)
[uncompress_layout](#)

create_module

Context: all contexts

Mode: insertion

Creates a new design module with no ports.

Usage

```
create_module module_name [-language {verilog | vhdl}] [-in_library library_name]
[-vhdl_architecture vhdl_architecture] [-silent]
```

Description

Creates a new design module with no ports.

The `create_module` command returns a collection of one module object where the module is a newly created module.

After creating a module with this command, you can set the new module as the top level of the design for all subsequent commands by using the [set_current_design](#) command and providing the name of the new module.

You can add ports to this new design module using the [create_port](#) command, add instances using the [create_instance](#) command, and then connect the pins using the [create_connections](#) command. Optionally, you can save the completed module to disk using the [write_design](#) command. See “[Design Editing Examples](#)” section of the *Tessent Shell User's Manual*.

After a new module is created, the current design can be reset to a different level with the `set_current_design` command and the module can be inserted with a `create_instance` command.

Arguments

- ***module_name***

A required string that specifies the name of the module to create.

- **-language verilog | vhdl**

An optional switch and literal pair that specifies which language (Verilog or VHDL) to use when creating the module. The language defaults to Verilog when this option is not specified. The `-language` option is only allowed in the `rtl` context.

- **-in_library *library_name***

An optional switch and string pair that specifies in which library to create the module. The specified `library_name` must be a library name previously defined with the [set_logical_design_libraries](#) command. When the `-in_library` option is not specified, the module is created inside the default library which is also specified using the [set_logical_design_libraries](#) command. The `-in_library` option is only allowed in the `rtl` context.

- **-vhdl_architecture** *vhdl_architecture*

An optional switch and string pair that specifies the name of the architecture to use for a created VHDL module. When **-vhdl_architecture** is not specified and **-language** is specified as vhdl, the created architecture is called struct. You cannot specify **-vhdl_architecture** unless **-language** is specified as vhdl. The **-vhdl_architecture** option is only allowed in the rtl context.

- **-silent**

An optional switch that suppresses the error that is normally generated if *module_name* already exists. A null collection is returned when an error is suppressed.

Examples

Example 1

The following example creates a module called mytop and makes it the current module so that other modules can be instantiated in it.

```
set_context dft -no_rtl
set_system_mode insertion
create_module mytop
{mytop}

set_current_design mytop
```

Example 2

The following example creates a module called myblock and appends the newly created module object to a collection referenced by the Tcl variable called *created_modules*.

```
append_to_collection created_modules [create_module myblock]
{myblock}
```

Example 3

The following example creates a VHDL module inside library lib1 with the default vhdl architecture struct. Notice how the **get_modules** command is used to report its attribute as **report_attribute** only looks for modules in the default library

```
create_module M1 -in_library lib1 -language vhdl
{M1}

report_attributes [get_modules M1 -in_library lib1]

Attribute Definition Report
```

```
Attributes defined for object 'M1':
Name          Value   Inheritance
-----
file_path_name      -
is_created        true    -
is_modified       false   -
is_unsaved        true    -
is_valid          true    -
language          vhdl   -
library_name       lib1   -
line_number        0      -
master_name        M1     -
name              M1     -
object_type        module  -
type              design  -
vhdl_architecture struc  -
```

Related Topics

[copy_module](#)
[create_connections](#)
[create_instance](#)
[create_port](#)
[get_modules](#)

create_net

Context: all contexts

Mode: insertion

Creates a net inside an instance of a design module in the current design.

Usage

```
create_net hier_net_name
    [-below_instance instance_object]
    [-ascending]
    [-silent]
```

Description

Creates a net inside an instance of a design module in the current design.

This command returns a collection of created net objects. This command returns one object for each bit of a bus with the left bit first and the right bit last.

Arguments

- ***hier_net_name***

A required string that specifies the path to the instance in which the net will be created, and the leaf name of the net to be created in the module. If the net name is a bus, this command creates one net for each bus bit, beginning with the left-most bit of the bus and ending with the right-most. The specified hierarchical path in which the net is to be created must exist in the design.

- **-below_instance *instance_object***

An optional switch and value pair that specifies to create the net below the specified *instance_object*. The tool searches the specified *name_patterns* relative to each instance found in the *instance_object* list. If the *instance_object* is a null collection or a null string, the switch is ignored and instances are searched relative to the current design.

- **-ascending**

An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- **-silent**

An optional argument that specifies the suppression of an error given one of the situations below. The command returns a null collection when an error is suppressed.

- The hierarchical path of the net name specified does not exist in the design
- The net already exist in the design

Examples

The following example creates a two-bit bus net called A inside instance u1.

create_net u1/A[4:5]

{u1/A[4] u1/A[5]}

Related Topics

[delete_pins](#)

[delete_nets](#)

[get_nets](#)

[get_icl_pins](#)

[get_icl_ports](#)

create_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Performs automatic high-speed test generation and pattern compression.

Usage

```
create_patterns [-OVERRIDE_user_settings | -NO_Auto]
[-Compaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
[-COverage_effort {Low | High}] [-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg] [-PATterns_per_pass integer]
[-NO_Report_statistics]
```

Description

Performs automatic high-speed test generation and pattern compression.

This command is the recommended way to produce high quality, efficient, test pattern sets in the shortest possible time.

This is a multiprocessing command for ATPG. For more information about multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

The create_patterns command uses the built-in expert system, ATPG Expert, to create patterns. ATPG Expert analyzes the design and DRC and may modify tool settings to optimize test patterns for coverage, pattern count, and run time. By default, ATPG Expert only optimizes settings that have not been explicitly specified; it does not change user-specified settings. If you want ATPG Expert to also optimize user-specified settings, use the -override_user_settings switch. Additionally, a statistics report is automatically displayed. The create_patterns command displays the following statistics:

- Number of patterns simulated
- Test coverage
- Number of faults detected
- Number of faults remaining in the fault list (undetected)
- Number of effective patterns
- Number of test patterns
- Number of redundant, untestable, and aborted faults

Occasionally, the display includes a line with three or four numbers separated by slashes (/). These numbers are, from left to right, the cumulative number of redundant (RE), ATPG_untestable (AU), and aborted faults. Aborted faults are separated into ATPG

aborts (AAB) and EDT aborts (EAB) depending on whether compression is on. The sum of AAB and EAB aborts equal the total number of aborts.

Example transcript without compression (these numbers are shown in bold font):

```
//#patterns test      #faults #faults    #eff.   #test      process    RE/AU/AAB
//simulated coverage in list detected patterns patterns CPU time
//  ---      ---      ---      ---      ---      ---      5.79 sec  0/40868/57
//  64      74.10%  135463  413980       63        63     14.73 sec
```

Example transcript with compression (these numbers are shown in bold font):

```
//#patterns  test      #faults #faults    #eff.   #test      process    RE/AU/AAB/EAB
//simulated coverage in list detected patterns patterns CPU time
//  ---      -----      ---      ---      ---      ---      5.79 sec  0/40868/27/30
//  64      74.10%  135463  413980       63        63     14.73 sec
```

Note

 At the start of create patterns, when the hardware has EDT and the EDT low-power controller is enabled, but [set_power_control](#) has not been used, the tool issues the following note. The note says “highest” because there may be multiple active EDT instances, and they may have different hardware thresholds.

```
// Note: Creating low-power patterns with <N>%
// shift switching threshold based on highest threshold of all active EDT
// low-power controllers or 15%, whichever is higher.
// To use a different low power shift threshold or disable low
// power shift, use the 'set_power_control shift on
// -switching_threshold_percentage <int>' command.
```

At the end of the `create_patterns` run, the tool automatically performs redundant fault identification, which is equivalent to running the [identify_redundant_faults](#) command. If, however, you set any ATPG limit (for example, test coverage limit, run time limit, or pattern limit), then the tool does not perform redundant fault identification.

You should use the `create_patterns` command, review the transcript for any suggested command or setting changes, and implement the ones that help you achieve your test goals. Be aware that some of the suggestions have trade-offs (for example, greater run time in exchange for fewer test patterns); the merits of the trade-offs depend on your particular test requirements.

Note

 When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, the tool assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

For static fault models, the tool automatically calls “add_faults -all” when the create_patterns command is executed and the fault list is empty.

For transition and UDFM with defect_delay fault models, the tool automatically calls “add_faults -clock_domains all” when the create_patterns command is executed and the fault list is empty. The tool does this to ensure that at-speed test is only performed within groups of synchronous clocks.

Note that you can use the [add_faults](#) command to define the scope of the create_patterns command. For more information, refer to “[Example 3](#)” on page 526.

The CPU time the tool reports is the time local to the command. If the command has multiple phases, then the reported CPU time is with respect to the beginning of that phase.

Arguments

- **-OVerride_user_settings**

An optional switch that enables ATPG Expert to override user-specified settings if necessary to create an optimal test pattern set.

- **-NO_AUto**

An optional switch that disables use of ATPG Expert to create patterns. During test generation, the default settings are used for those settings that have not been explicitly specified.

- **-Compaction_effort {Low | High | Maximum | *integer*}**

An optional switch that specifies how much effort to spend trying to reduce the pattern count:

Low

Specifies a faster run time but a higher pattern count than the High option.

High

Specifies a longer run time but a smaller pattern count than the Low option. Use High when generating production patterns because this can significantly reduce test time and test data volume. This option usually gives the best results and is the default.

Maximum

Enables maximum tool effort to reduce pattern count. This option significantly increases run time and should be used only when absolutely necessary. Integer value 10 or 100 should be used before using this option due to the significantly long run time.

integer

A positive integer value that specifies compaction effort as follows:

0 = Low

1 = High (usually give the best results)

n = The tool works n times harder than High

As a general guide, when $n=10$, the run time increase is small, and the pattern count can be slightly better. When $n=100$, pattern reduction can be similar to Maximum, while the run time can be lower than Maximum. Differences in design complexity and size can affect which integer value provides the desired trade-off between pattern counts and run time.

Note that you can significantly reduce the run time caused by higher integer values by using multiprocessing. For more information about multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

- -NO_Analyze

An optional switch that skips the analysis of test generation problems. Normally, before test generation, `create_patterns` checks for bus contention and whether ATPG restrictions and constraints can be satisfied. If these checks reveal potential test generation problems, an error message displays and the command terminates. If you want the analysis to be ignored and to proceed with test generation, reissue the command with this switch.

Note

 Because this switch prevents normal analysis, all faults can be targeted, and ATPG effort expanded, yet without generating patterns. The extra effort may also give the appearance that the tool is stuck. Use this switch only in special cases when the risks are acceptable.

- -COverage_effort {Low | High}

An optional switch that improves ATPG test coverage. When set to High, the tool retargets aborted faults and uses dynamic learning during retargeting. Note that using this setting can result in significantly higher run times in some cases. By default, the tool does not retarget aborted faults.

Using this switch may increase the ATPG run time.

- **-NO_REDundancy_analysis**

An optional switch that omits the redundancy identification analysis that follows pattern generation. When you use this switch, the patterns generate faster, but the test coverage calculations are less accurate.

- **-NO_TERMINATE_ineffective_atpg**

An optional switch that disables a feature that terminates the ATPG session after the patterns are no longer efficient at detecting faults. Note that using this switch results in a longer run time and may generate many additional patterns to detect very few faults.

- **-PATterns_per_pass *integer***

An optional switch and positive integer pair that specifies how many patterns to create and simulate in each pass. The integer value must be between 1 and 64. By default, the tool creates and simulates 64 patterns in each pass. When the value is 1, the tool generates and simulates every pattern, and so on. The switch affects only the current `create_patterns` session.

The potential benefit of using a smaller integer value rather than the default 64 is better pattern count because ATPG is less likely to add non-essential faults into a pattern.

However, when the value is small, the overall run time is likely to increase due to excessive fault simulation. Furthermore, a small value can reduce the run-time efficiency of multithreading and distributed ATPG.

A smaller value is appropriate if the expected pattern count is small. A typical scenario for using a small integer value is IDQ pattern generation, where you should use “`create_patterns -patterns_per_pass 1`” with “`set_atpg_limits -pattern_count n`” for creating *n* IDQ patterns. You can use the `-compaction_effort` option with `-patterns_per_pass` to maximize the effectiveness of each pattern, but at the expense of additional run time.

- **-NO_REPort_statistics**

An optional switch that suppresses statistics reporting after the tool creates patterns. This is helpful for avoiding duplicate reports when you want to run the `report_statistics` command with non-default options immediately after running `create_patterns`.

Examples

Example 1

The following example creates, through ATPG Expert, an internally-stored set of optimally compact patterns for the entire netlist that can be saved using the `write_patterns` command. ATPG Expert automatically decides the best setup (for example, sequential depth) for the design.

```
set_system_mode analysis
create_patterns
```

Example 2

The following example also creates an internally-stored set of compact patterns; however, ATPG Expert has been disabled. In this case, you need to specify by which sequential depth the

create_patterns

pattern is created. The -sequential 2 switch means there are up to two clock pulses between scan loads. The patterns can be saved using the write_patterns command.

```
set_system_mode analysis
set_pattern_type -sequential 2
create_patterns -no_auto
```

Example 3

The following example creates patterns only for the specific block specified by the add_faults command:

```
set_system_mode analysis
add_faults /blockA/blockB
create_patterns
```

Example 4

The following example creates, through ATPG Expert, an internally-stored set of optimally compact at-speed fault patterns for the entire netlist that can be saved using the write_patterns command. Notice that for at-speed fault models, if you are not using named capture procedures, and there are no user defined settings or you are not calling create_patterns with the -override switch, the tool will automatically turn on hold_pi and mask all primary outputs. It will also automatically set clock restriction to "domain_clock" with any_interaction and compatible_clocks_between_loads ON.

```
set_system_mode analysis
set_fault_type transition
rep_faults

// Warning: No faults reported.

create_patterns
```

```

// | -----
// |           Analyzing the design
//
// | Current clock restriction setting: Domain_clock (edge interaction)
// | Calling: set_clock_restriction domain_clock -any_interaction
// |           -compatible_clocks_between_loads on
//
// | Current split capture setting:          Off (optimal)
//
// | Current clock off simulation setting: On(optimal)
//
// |           Current holdpi setting:      Off
// |           Calling:                  set_transition_holdpi on
//
// |           Current output masks setting: Off
// |           Calling:                  set_output_masks on
//
// |           Current abort limit setting: 300 (Combinational) 100 (Sequential)
// | -----
//
// |           Current sequential depth:    2 (optimal)
//
// | -----
// |
// | -----  

// | Simulation performed for #gates = 654 #faults = 2497
// | system mode = ATPG      pattern source = internal patterns
// | -----
// | #patterns test      #faults #faults      # eff.      # test      process      RE/AU/AAB
// | simulated coverage in list detected      patterns      patterns      CPU time
// | deterministic ATPG invoked with comb/seq abort limit = 300/100
// | ---      -----      ---      ---      ---      ---      ---      0.30 sec      21/15/5
// | 64       79.86%     533      1949      59       59       0.31 sec
// | ---      -----      ---      ---      ---      ---      ---      0.54 sec      21/91/19
// | 128      89.31%     200      257       54      113      0.54 sec
// | ---      -----      ---      ---      ---      ---      ---      0.64 sec      21/173/29
// | 190      92.80%     23       95       49      162      0.64 sec

```

```

Statistics Report
Transition Faults
-----
Fault Classes          #faults
                        (total)
-----
FU (full)           2758
-----
UO (unobserved)    23 ( 0.83%)
DS (det_simulation) 2301 (83.43%)
DI (det_implementation) 224 ( 8.12%)
UU (unused)         4 ( 0.15%)
TI (tied)           12 ( 0.44%)
RE (redundant)      21 ( 0.76%)
AU (atpg_untestable) 173 ( 6.27%)
-----
Fault Sub-classes
-----
AU (atpg_untestable)
  SEQ (sequential_depth) 44 ( 1.60%)
  Unclassified           129 ( 4.68%)
UC+UO
  AAB (atpg_abort)     23 ( 0.83%)
-----
Coverage
-----
test_coverage        92.80%
fault_coverage       91.55%
atpg_effectiveness  99.17%
-----
#test_patterns        162
#basic_patterns        2
#clock_sequential_patterns 160
#simulated_patterns   190
CPU_time (secs)      1.4
-----
```

Related Topics

[add_faults](#)
[identify_redundant_faults](#)
[report_statistics](#)
[reset_state](#)
[set_abort_limit](#)
[set_atpg_limits](#)
[set_fault_type](#)
[set_pattern_type](#)
[set_random_atpg](#)
[simulate_patterns](#)

[write_patterns](#)

create_patterns_specification

Context: All context

Modes: All modes

Prerequisite: The top level ICL module must exist and be elaborated.

Generates a pattern specification for the specified usage. The usage is either signoff or manufacturing.

Usage

```
create_patterns_specification [usage] [-replace] [-bsdl_files bsdl_file]
```

Description

Generates a pattern specification for the specified usage. The usage is either signoff or manufacturing. Currently the tool support IJTAG and MemoryBIST patterns.

This command can be called in any context and any system mode as long as the ICL module associated with the top-level design is available and elaborated, or if a BSDL file is specified for a “BSDL-only” design flow. You can provide a top-level ICL module description by using first the [read_icl](#) command followed by the [set_current_design](#) command, or by performing ICL extraction using the [extract_icl](#) command. The context will automatically be changed to patterns if called from a different context.

This command returns the name of the [PatternsSpecification](#) wrapper if it was just created in memory. This makes it easier to modify using the configuration data editing and introspection commands listed in [Table 10-1](#) on page 3175 as shown in the example below.

Note

 The DftControlSettings and the [ProcedureStep](#) wrapper are auto filled by the command. See the [Patterns](#) section for a description of where the DftControlSettings list comes from. See the documentation of the [tessent_default_procedure_name](#) and [tessent_default_procedure_parameters](#) attributes for a description of their influence on the ProcedureStep inference.

Arguments

- *usage*

Specifies the intended usage of the patterns. The allowed values are as follows:

signoff — Indicates that simulation signoff patterns are requested. Signoff patterns are patterns that are used for simulation and verification of the design. They only include a sub-set of patterns by default. With this usage, the ICL file is analyzed to obtain the list of instruments and BIST controllers present in the design. A set of [Patterns](#) wrapper is created per instrument type to fully verify their proper operation inside the design. The properties found inside the [SignOffOptions](#) wrapper are used to configure how the patterns are created. By default, only the instruments and the BIST

controllers found in the top physical instance are simulated. You can set the “simulate_instruments_in_lower_physical_instances” property inside the [SignOffOptions](#) wrapper to also request that the instruments and the BIST controllers found in the lower physical instance be re-simulated from the current level. To help with simulation time, those simulation patterns are created so as to have one [Patterns](#) wrapper per instrument type/per lower physical instance which enables the simulation to be done using only the full view of one lower physical instance and an empty view for the rest. See the description of the [LowerPhysicalBlockInstances](#) wrapper found inside the [SimulationOptions](#) wrapper as well as the description of the *design_name.format_interface* file.

In the signoff simulation, the memory BIST controllers (including ROM BIST controllers) are simulated using the [MemoryBist/reduced_address_count](#) property set to On. This assumes that the memory library description has been generated by a certified memory library generator and is correct by construction. If the memory library was created or modified manually, you must simulate the full address space on the memory BIST assembly module, located in the [instruments](#) directory, using the flow mentioned below. Because the top module name ends with “_mbist_assembly”, the [create_patterns_specification](#) command knows to not set the [MemoryBist/reduced_address_count](#) property to “on” so as to simulate the full address space of the memory.

```
> set_context patterns
> read_icl path/design_name_design_id_mbist_assembly.icl
> set_current_design design_name_design_id_mbist_assembly
> create_patterns_specification
> process_patterns_specification
> run_testbench_simulations
> check_testbench_simulations
```

manufacturing — Indicates the manufacturing patterns are requested. Manufacturing patterns are patterns generated for the ATE. They are a complete set of patterns by default. With this usage, the ICL file is analyzed to obtain the list of instruments and BIST controllers present in the design. A set of [Patterns](#) wrappers is created per instrument type to properly exercise the various test features during manufacturing test. The properties found inside the [ManufacturingOptions](#) wrapper are used to configure how the patterns are created. Based on the [max_async_clock_sources](#) property value, the patterns will be organized to optimize parallelism without exceeding the amount of independent frequency clocks your testcase can supply. Unlike with the original LV flow where you could only have 0 or an unlimited amount of asynchronous clocks, you can know and precisely specify the exact amount of asynchronous clocks available on your tester; the [PatternsSpecification](#) will be optimized for that constraint.

- -replace

With this option, the error that would normally be generated is ignored when running the command where the [PatternsSpecification](#) wrapper with the same *design_name* and *design_id* and the *pattern_id* already exists in memory. This allows the [PatternsSpecification](#) wrapper present in memory to be replaced with the new one.

- **-bsdl_files bsdl_file**

An optional value pair that specifies the BSDL file list to be used in creating the patterns specification for a “BSDL-only” design flow. If this argument is not specified, the BSDL file specified with the [read_icl-bsdl_file](#) option will be used. Refer to the “[BSDL-Only Flow](#)” section of the *Tessent BoundaryScan User’s Manual* for more information.

Examples

The following example calls the `create_patterns_specification` command and uses the configuration editing commands to insert in all memory BIST pattern wrappers a `DftControlSetting` wrapper above the first `TestStep` wrapper that includes a `DEBUG` macro definition.

```
> set spec [create_patterns_specification]
> set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
> foreach_in_col mbist_pat $mbist_pats {
>   set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
>   read_config_data -before $first_test_step -from_string {
      SimulationOptions {
        SimulationMacros {
          DEBUG;
        }
      }
}
}
```

Related Topics

[apply_specification_defaults](#)
[process_patterns_specification](#)

create_pin

Context: all contexts

Mode: insertion

Creates a pin on an instance of a design module.

Usage

```
create_pin hierPinName [-direction input | output | inout] [-ascending] [-silent]
```

Description

Creates a pin on an instance of a design module.

If –auto_uniquify_edited_modules is On, the module is first uniquified before creating a pin on the specified instance. If –auto_uniquify_edited_modules is Off, the pin is created on the module and all instances of the module get the newly created pin.

This command returns a collection of the pin objects created; if hierPinName is a bus, this command returns a collection with one pin object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

Arguments

- ***hierPinName***

A required string that specifies the hierarchical path of the pin to be created concatenated to the leaf pin name. The specified hierarchical path of the pin to be created must exist in the design.

- **-direction input | output | inout**

An optional switch and literal pair that specifies the direction of the newly created pin. An input pin is created by default when this option is omitted.

- **-ascending**

An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- **-silent**

An optional argument that specifies to not generate an error message if the specified hierPinName already exists or when the hierarchical part of the specified name is not an existing instance within the current design. When an error is suppressed, the command returns an empty collection.

Examples

The following example creates a 5-bit bus input pin called A on the instance u1. The command returns a collection of newly created pin objects.

```
create_pin u1/A[5:1]
{u1/A[5] u1/A[4] u1/A[3] u1/A[2] u1/A[1]}
```

The following example introspects the direction attribute of the four newly created pin objects.

```
get_attribute_value_list [create_pin u1/C[3:0] -direction output] -name direction
{output output output output}
```

Related Topics

[delete_pins](#)

[get_nets](#)

[get_icl_ports](#)

[get_icl_pins](#)

create_port

Context: dft

Mode: insertion

Creates a port on the specified design module.

Usage

```
create_port port_name [-on_module obj_spec] [-direction input | output | inout]  
[-ascending] [-silent]
```

Description

Creates a port on the specified design module.

By default, this command creates a port on the design module in the current design. If you specify the -on_module argument, this command creates a port on the module specified in obj_spec.

This command returns a collection of the port objects created; if port_name is a bus, this command returns a collection with one port object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

If the specified port already exists the tool generates an error. You can specify the -silent switch to instruct the tool not to generate an error.

Arguments

- ***port_name***
A required value that specifies the name of the port to be created. Bus ports must be created with a single create_port command where port_name includes the required bus range.
- **-on_module *obj_spec***
An optional switch and value pair that specifies the module to which the port is to be added. The obj_spec can be the name of a module or a module object returned by the get_module command. If the -on_module switch is not specified, this command adds the port to the current design which is the module specified by the set_current_design command. You cannot create a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to create ports on modules created with the create_module command.
- **-direction input | output | inout**
An optional switch and literal pair that specify the direction of the newly created port. An input port is created when this option is omitted.
- **-ascending**
An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right

index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- -silent

An optional argument that specifies to not generate an error message if the specified port_name already exists. When an error is suppressed, the command returns an empty collection.

Examples

Example 1

The following example creates a 5-bit bus input port called A on the module that is the current design. The command returns a collection of the newly created port objects.

```
create_port A[5:1]
```

```
{A[5] A[4] A[3] A[2] A[1]}
```

Example 2

The following example creates a scalar output port called B on module ModB.

```
create_port B -on_module ModB -direction output
```

```
{B}
```

Example 3

The following example creates a 4-bit bus output port called C on the current design and uses the get_attribute_value_list command to query the direction attribute of the newly created port objects.

```
get_attribute_value_list [create_port C[3:0]] -name direction
```

```
{input input input input}
```

Related Topics

[delete_ports](#)

[get_icl_ports](#)

create_scan_chain_family

Context: dft -scan

Mode: analysis

Creates a scan_chain_family object, which controls how one or more scan chains will be allocated/connected during distribution.

Usage

```
create_scan_chain_family family_name
  { -include_elements obj_spec | -ordered_elements obj_spec }
  [ -edt_instances obj_spec | { -si_connections obj_spec -so_connections obj_spec } ]
  [ -si_associated_ports obj_spec -so_associated_ports obj_spec ]
  [ -single_class_chains { on | off } ]
  [ -single_clock_domain_chains { on | off } ]
  [ -single_clock_edge_chains { on | off } ]
  [ -single_power_domain_chains { on | off } ]
  [ -single_cluster_chains { on | off } ]
  [ -port_index_start_value integer ]
  [ -port_scalar_index_modifier integer ]
  [ -chain_length { integer | unlimited } ]
  [ -chain_count { integer | unlimited } ]
  [ -si_port_format format ]
  [ -so_port_format format ]
  [ -si_timing { leading_edge | trailing_edge | any_edge } ]
  [ -so_timing { leading_edge | trailing_edge | any_edge } ]
  [ -single_wrapper_type_chains { On | Off } ]
  [ -lockup_cell_type { latch | flop } ]
  [ -si_lockup_cell_type { latch | flop } ]
  [ -so_lockup_cell_type { latch | flop } ]
  [ -unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input } ]
  [ -si_pipelining { wrapper_chains_only | all_chains | off } ]
  [ -si_pipelining_type { non_scan_flop | holding_scan_flop } ]
```

Description

Use this command to create a scan_chain_family object, which controls how one or more scan chains will be allocated/connected during distribution. This is an optional/advanced command that enables you to fine-tune how sub-populations of a mode get treated.

Arguments

- ***family_name***

A required string that specifies a scan chain family name.

- **-include_elements *obj_spec***
A required switch (if -ordered_elements is not used) and value pair that specifies a list of mandatory scan elements to include in this chain family. It has no default value.
- **-ordered_elements *obj_spec***
A required switch (if -include_elements is not used) and value pair that specifies an ordered list of mandatory scan elements to include in this chain family. It has no default value.
- **-edt_instances *obj_spec***
A switch and value pair that specifies EDT instances to connect the scan chains to. When core descriptions for the specified instances are detected, then the tool automatically sets the -si/ so_connections to the instances scan pins.
- **-si/so_connections *obj_spec***
A switch and value pair that specifies connections to existing SI/SO pins or ports. The specified value can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). The specified values corresponds to existing top-level input port(s) or output pin(s) for -si_connections and existing top-level output port(s) or input pin(s) for -so_connections.
- **-si_associated_ports *obj_spec* -so_associated_ports *obj_spec***
Optional string and literal pairs that specify existing top-level port(s) associated to the internal connections identified using -si/so_connections. These are used in the .tcd file to describe the scan ports of the chains at the core boundary.
- **-single_class_chains { On | Off }**
An optional string and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. By default, the tool will keep wrapper cells separate from non-wrapper chains for internal and external type of scan modes. However, for unwrapped mode they can be mixed.
- **-single_clock_domain_chains { On | Off }**
An optional string and literal pair that specifies whether to restrict a scan chain to a single clock domain. The default is off.
- **-single_clock_edge_chains { On | Off }**
An optional string and literal pair that specifies whether to restrict a scan chain to cells having a single edge. The default is off.
- **-single_power_domain_chains { On | Off }**
An optional string and literal pair that specifies to restrict a scan chain to a single power domain using the attribute: power_domain_name. The default is off.
- **-single_cluster_chains { On | Off }**
An optional string and literal pair that specifies to restrict a scan chain to a single cluster using the attribute: cluster_name. The default is on.

- **-port_index_start_value *integer***
An optional switch and integer that specifies the starting port index. The starting index is 0 by default, but can be overridden by this option.
- **-port_scalar_index_modifier *integer***
An optional switch and integer that specifies the increment used between each chain port indices, the default is 1.
- **-chain_length {*integer* | unlimited }**
An optional integer that specifies the target length of scan chains for a particular chain family. By default, the no limit (unlimited) is imposed on the chain length. The minimum scan chain length is 1.
- **-chain_count { *integer* | unlimited }**
An optional integer that specifies the number of scan chains that should be constructed for the specified chain family. By default, the chain count is set to the number of connections specified with the **-si/so_connections** or by the number of pins of the EDT controller(s) specified with the **-edt_instances** option, otherwise if the latter options were not used, then no limit (unlimited) is imposed on the chain count.
- **-si/so_port_format *format***
An optional string that specifies the port format. The default si/so_port_format is set to “ts_%s_si/so[%d]” when there are more than one scan mode, otherwise it defaults to “ts_si/so[%d]”. The mode placeholder “%s” gets replaced by the chain mode name, whereas the index placeholder “%d” gets substituted by the chain index. A precision number may be used when specifying the index placeholder, for example the format “lsi_si%2d” would cause port names such as “lsi_si00”, “lsi_si01” to be created. Note that leading zeros are inserted automatically when needed making “%2d” implicitly equivalent to “%02d”.
- **-si_timing { leading_edge | trailing_edge | any_edge }**
An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in the chain to control the capture edge of the first cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is leading_edge.
- **-so_timing { leading_edge | trailing_edge | any_edge }**
An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in the chain to control the change edge of the last cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is “trailing_edge”.
- **-lockup_cell_type { latch | flop }**
An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch

- **-si_lockup_cell_type { latch | flop }**

An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch

- **-so_lockup_cell_type { latch | flop }**

An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

Note



By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

- **-unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input }**

An optional switch that selects how to handle unused EDT chain pins. Options are:

insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.

tie_compactor_input — ties-off the EDT compactor.

auto — selects “insert_pipelining_flop” for EDT controllers with bypass chains, and “tie_compactor_input” for all other EDT controllers. This is the default.

- **-si_pipelining { wrapper_chains_only | all_chains | off }**

An optional switch and string that controls whether a pipelining flop gets inserted at the beginning of chains. When set to all, all scan chains get a pipelining flop. When set to wrapper_chains_only, then only chains of internal or external mode types and composed of wrapper scan elements get a pipelining flop. The default is wrapper_chains_only

- **-si_pipelining_type {non_scan_flop | holding_scan_flop}**

An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

create_silicon_insight_setup_specification

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Generates a default Tesson SiliconInsight Setup Specification.

Usage

`create_silicon_insight_setup_specification`

Description

Generates a default Tesson SiliconInsight Setup Specification wrapper called `SiliconInsightSetupSpec` and saves it to the current working directory in a file named `SiliconInsightSetupSpec.cfg`.

The TSI setup spec contains Tesson SiliconInsight-specific data that define the pin map, testers, adaptor types, or other data that are required to prepare Tesson SiliconInsight to communicate with the DUT, simulator process or ATE.

For information about the Tesson SiliconInsight setup specification, refer to “[Prepare the Design Under Test](#)” in the *Tesson SiliconInsight User’s Manual for Tesson Shell*.

Arguments

None

delete_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes state restrictions from the specified pins. You define pin restrictions by using the add_atpg_constraints command.

Usage

```
delete_atpg_constraints {pin.pathname | gate_ID# | function_name |  
 {-Cell cell_name {pin_name...}}... [-Instance {object_expression...}]  
 [-MODULE {module_name ...}]... | -All
```

Description

Removes state restrictions from the specified pins. You define pin restrictions by using the [add_atpg_constraints](#) command.

If you change ATPG constraints, create patterns with those changed constraints, and then compress patterns, all patterns not meeting the new constraints are rejected. This can cause the good patterns created with the old constraints to be rejected; therefore, use the delete_atpg_constraints command to remove all ATPG constraints before compressing the pattern set.

Arguments

- ***pin.pathname***

A repeatable string specifying the pathname of the pin from which to remove any ATPG pin constraints.

- ***gate_ID#***

A repeatable integer specifying the gate identification number of the gate from which to remove any ATPG pin constraints.

- ***function_name***

A repeatable string specifying the name of a function created with the add_atpg_functions command and from which to remove any ATPG pin constraints.

- **-Cell *cell_name* {*pin_name*}**

A repeatable switch and string pair that removes ATPG constraints from the specified pin on the DFT library cell. You can repeat the *pin_name* argument if there are multiple pins or nets on a cell that have ATPG constraints you need to remove.

- **-Instance *object_expression***

An optional switch and repeatable string pair that removes ATPG constraints inside the specified list of instances. You can use regular expressions, which may include any number of embedded asterisk (*) and/or question mark (?) wildcard characters. You can only use this switch when using the -Cell option or when a function name is specified.

- **-MODULE *module_name***

An optional switch and repeatable string pair that removes the ATPG constraints inside all instances of the specified module.

- **-All**

A switch that removes all current, user-defined ATPG constraints from all objects.

Examples

Example 1

The following example creates two user-defined ATPG pin constraints, runs the ATPG process, removes all ATPG constraints, and then compresses the pattern set:

```
set_system_mode analysis
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
create_patterns
delete_atpg_constraints -all
compress_patterns
```

Example 2

The following Tesson FastScan example shows how to use wildcards.

```
add_atpg_constraints 1 -cell mux21 Y -ins uALU/*/ix184
report_atpg_constraints

1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
1 dynamic /uALU/u1/ix184/Y (4412) .

delete_atpg_constraints -cell mux21 Y -ins uALU/u1/*
report_atpg_constraints

1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
```

Related Topics

[add_atpg_constraints](#)
[add_atpg_functions](#)
[report_atpg_constraints](#)

delete_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes the specified function definitions.

Usage

`delete_atpg_functions function_name... | -All`

Description

Removes the specified function definitions.

The `delete_atpg_functions` command lets you `delete_atpg_functions` defined with the [add_atpg_functions](#) command. You cannot remove an ATPG function if an ATPG constraint is currently using that function. If you attempt to remove an in-use function, the tool generates an error. Therefore, if you need to delete an in-use ATPG function, you must first remove all the associated ATPG constraints using the `delete_atpg_constraints` command; then you can remove the ATPG function.

You can display a list of the current ATPG functions that the tool is using as ATPG constraints by using the `report_atpg_constraints` command.

Arguments

- ***function_name***
A repeatable string that specifies the names of the ATPG functions that you want to delete.
- **-All**
A switch that removes all ATPG function definitions.

Examples

The following example creates two user-defined ATPG functions, one user-defined ATPG constraint, displays the currently-in-use ATPG constraints, and then removes one of the inactive ATPG functions:

```
add_atpg_functions and_b_in And /i$144/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
add_atpg_constraints 0 /i$135/q
report_atpg_constraints

0 /$135/Q (23)

delete_atpg_functions and_b_in
```

Related Topics

[add_atpg_functions](#)

[delete_atpg_constraints](#)

[report_atpg_constraints](#)

[report_atpg_functions](#)

delete_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.

Usage

`delete_bist_capture_ranges capture_proc_name | -All`

Description

Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.

Arguments

- ***capture_proc_name***
A required string that specifies the name of the capture procedure.
- **-All**
A switch that specifies that the tool no longer use any named capture procedures when generating BIST patterns.

Related Topics

[add_bist_capture_range](#)

[report_bist_capture_ranges](#)

delete_black_boxes

Context: all contexts

Mode: setup

Undoes the effect of the [add_black_boxes](#) command.

Usage

```
delete_black_boxes -Instances [ins_pathnames] | -Modules [module_names] | -All
```

Description

Undoes the effect of the [add_black_boxes](#) command.

For a module that was originally modeled, removing the effect of the [add_black_boxes](#) command reinstates the original model. Specified tied values are no longer set on the output pins. For a module that was empty or undefined in the input netlist, the output pins revert to the default tied value of X.

Note

 The tool releases the flat model if one exists at the time you issue this command.

Arguments

- **-Instances** [*ins_pathnames*]

A switch that specifies for the tool to undo the effect of the [add_black_boxes](#) command on all instances specified by the *ins_pathnames* argument. The *ins_pathnames* argument can be a Tcl list of one or more instances, or a collection of one or more instances. If you do not specify the *ins_pathname*, the tool undoes the effect of the [add_black_boxes](#) command on all instance-based blackboxes.

- **-Modules** [*module_names*]

A required switch that specifies for the tool to undo the effect of the [add_black_boxes](#) command on all instances of the Verilog module or Tessent Cell library model specified by the *module_names* argument. The *module_names* argument can be a Tcl list of one or more modules, or a collection of one or more modules. If you do not specify the *module_names* argument, the tool undoes the effect of the [add_black_boxes](#) command on all module-based blackboxes.

- **-All**

A required switch that specifies for the tool to undo the effect of the [add_black_boxes](#) command on all blackboxes.

Examples

Example 1

The following example adds the black box for module “core” then undoes all blackboxes that were defined.

```
add_black_boxes -modules core 1  
delete_black_boxes -all
```

Example 2

The following example deletes blackboxes for the collection of modules returned by the get_modules command. The get_modules command returns only those modules that have been marked by the user with the user-defined attribute mark_black_box.

```
delete_black_boxes -modules [get_modules -filter mark_black_box]
```

Related Topics

[add_black_boxes](#)

[report_black_boxes](#)

delete_browser_data

Context: all contexts

Mode: setup, analysis

Removes data from the active tab of the DFTVisualizer Browser window.

Usage

```
delete_browser_data {Gates | Primitives | {FAults [-All | -Total | fault_class] } |  
    F_coverage | T_coverage | Relevant_Test_Coverage | Loss_TC | ATpg_effectiveness |  
    All } [-Tab {INstance | Library | Clock}]
```

Description

Removes data from the active tab of the DFTVisualizer Browser window.

You use the *-Tab* switch to specify to remove the data column from a tab that is not the active tab.

Multiple arguments can be used in one `delete_browser_data` command.

Arguments

- **Gates**

Required literal that deletes the design level gates listed for each instance.

- **Primitives**

Required literal that deletes the primitive level gates listed for each instance.

- **FAults**

Required repeatable literal that deletes fault information from the Browser window. By default all fault information is deleted. This switch is valid in the Hierarchy and Clocks tabs of the Browser window. Options include:

- Total

Switch that deletes the display of total faults for all classes.

- fault_class*

String that specifies a *fault_class* to delete faults for. The *fault_class* must be the character string shown at the top of the column to delete. For a summary of available fault classes, refer to the `report_faults` command.

- All

Switch that specifies to delete all fault classes that are currently displayed.

- **F_coverage**

Required literal that deletes all the fault coverage data from the display.

- **T_coverage**

Required literal that deletes all the test coverage data from the display.

- **Relevant_Test_Coverage**

Required literal that deletes the test coverage for each submodule after untestable faults have been added/deleted from test coverage calculations.

- **Loss_TC**

Required literal that deletes the undetected faults in an instance displayed as a percentage of the testable faults in the entire design.

- **ATpg_effectiveness**

Required literal that deletes all the ATPG effectiveness data from the display.

- **ALI**

Required switch that deletes all data from the display.

- **-Tab {Instance | Library | Clock}**

Optional switch and literal that specifies the tab in the Design Browser window from which the column is to be deleted. By default, the delete_browser_data command removes data from the active window. You can use this switch to specify to remove the data column from a tab that is not the active tab.

Examples

The following example adds columns providing the fault statistics for each instance, and then removes the column that shows the Detected by Implication (DI) fault statistics:

```
add_browser_data primitives faults -all  
delete_browser_data primitives faults di
```

Related Topics

[add_browser_data](#)

delete_capture_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes the special data capture handling for the specified objects.

Usage

```
delete_capture_handling {object... | -All} [-SInk | -SOurce]
```

Description

Removes the special data capture handling for the specified objects.

When you remove the special data capture handling, the default handling resumes. The default data capture handling specifies that the source elements pass on the values from the previous (not the current) clock cycle. When using the delete_capture_handling command, you must specify either an object name or all objects.

Arguments

- ***object***

Specifies the object(s) for which you want the tool to remove any special data capture handling. The following lists the valid choices for the *object* argument:

gate_id# — A repeatable integer that specifies the gate identification numbers of the objects. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

pin.pathname — A repeatable string that specifies the name of a pin within the design.

instance_name — A repeatable string that specifies the name of an instance within the design.

-Cell *cell_type* — A repeatable switch and string pair that specifies the name of a cell.

- **-All**

A switch that removes all special data capture handling.

- **-SInk**

An optional switch specifying that the *object* argument is a termination point. This is the default behavior for the command.

If you use this switch in combination with the -All switch, the tool removes all special data capture handling on all the sink elements.

- **-SOurce**

An optional switch specifying that the *object* argument is an origination point for data capture.

If you use this switch in combination with the -All switch, the tool removes all special data capture handling on all the source elements.

Examples

The following example changes the data capture handling of two specific gates and then removes one of those changes.

```
add_capture_handling new 1158 1485 -source  
delete_capture_handling 1158
```

Related Topics

[add_capture_handling](#)
[report_capture_handling](#)

delete_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Removes the specified capture procedures from internal memory.

Usage

```
delete_capture_procedures {-All | -Created_capture_procedures |
    -Loaded_capture_procedures | -Disabled | procedure_name...}
```

Description

Removes the specified capture procedures from internal memory.

The delete_capture_procedures command deletes all or specific named capture procedures from the internal tool memory. A named capture procedure used by the current pattern set cannot be deleted.

Arguments

- **-All**
A switch that deletes all the named capture procedures.
- **-Created_capture_procedures**
A switch that deletes all the named capture procedures automatically created by the tool.
- **-Loaded_capture_procedures**
A switch that deletes all the named capture procedures loaded using the [read_profile](#) command or from procedure files associated with the scan group definition.
- **-Disabled**
A switch that deletes all the disabled named capture procedures.
- **procedure_name**
A repeatable string that specifies a named capture procedure to delete.

Examples

Example 1

The following example deletes all capture procedures.

```
delete_capture_procedures -All
```

Example 2

The following example removes the capture procedure named *user_capture_proc1*.

```
delete_capture_procedures user_capture_proc1
```

Related Topics

[create_capture_procedures](#)
[report_capture_procedures](#)
[set_capture_procedures](#)

delete_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Removes the specified test from the CDP.

Usage

`delete_cdp_test test_name`

Arguments

- *test_name*

A required string that specifies the name of a CDP test.

Examples

The following example removes the test1 CDP test:

`delete_cdp_test test1`

Related Topics

[add_cdp_test](#)

delete_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes constraints placed on scan cells.

Usage

```
delete_cell_constraints -All | pin.pathname | {chain_name cell_position} | {-Drc C6} |  
{-CHain chain_name} | {-Clock clock_name}
```

Description

Removes constraints placed on scan cells.

The delete_cell_constraints command deletes the constraints placed on scan cells using the [add_cell_constraints](#) command. You can specify a scan cell by using either a pin pathname or a position in a scan chain.

Arguments

- **-All**
A switch that, in Setup mode, specifies to delete all constraints from all scan cells. In analysis mode, it specifies to delete all cell constraints that were added in non-Setup modes.
- ***pin.pathname***
A string that specifies the name of an output pin which directly connects to a scan memory element (you can only specify output pins of buffers and inverters). Valid in Setup mode only.
- ***chain_name cell_position***
A string and integer pair that specifies the name of a currently-defined scan chain and the position of the cell in the scan chain. The *cell_position* is an integer where 0 is the scan cell closest to the scan-out pin. Valid in Setup mode only.
- **-Drc C6**
A switch and literal pair that specifies to delete all cell constraints added by “add_cell_constraints -drc c6”. Valid in analysis mode only.
- **-CHain *chain_name***
A switch and string pair that specifies to delete all cell constraints added by the -Chain argument to the add_cell_constraints command. Valid in analysis mode only.
- **-Clock *clock_name***
A switch and string pair that specifies to delete all cell constraints added by the -Clock argument to the add_cell_constraints command. Valid in analysis mode only.

Examples

The following example deletes an incorrectly added cell constraint placed on a scan cell:

```
add_clocks 1 clock1
add_scan_groups group1 proc.g1
add_scan_chains chain1 group1 scanin1 scanout1
add_scan_chains chain2 group1 scanin2 scanout2
add_cell_constraints chain1 5 c0
add_cell_constraints chain2 3 c1
delete_cell_constraints chain2 3
add_cell_constraints chain2 4 c1
report_cell_constraints
```

Related Topics

[add_cell_constraints](#)

[report_cell_constraints](#)

delete_cell_library

Context: unspecified, all contexts

Mode: setup

Removes all current cell libraries, the current design including the flat model, and all tool data that is design-specific.

Usage

`delete_cell_library`

Description

Removes all current cell libraries, the current design including the flat model, and all tool data that is design-specific.

This command also closes any open DFTVisualizer windows.

When this command is deleting a flat model, and that flat model was saved with a list of enabled/disabled commands, then the restriction on the disabled commands is removed. In other words, all commands allowed in the tool will be enabled for the next design.

Arguments

None

Examples

The following example removes all current cell libraries:

`delete_cell_library`

Related Topics

[delete_design](#)

[read_cell_library](#)

[read_flat_model](#)

[read_verilog](#)

[set_current_design](#)

[set_design_macros](#)

delete_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.

Usage

```
delete_cell_models dftlib_model... | {-Type {INV | And | Buf | OR | NAnd | NOR | Xor | Mux  
| Scancell | DFF | LATCH}} | -All
```

Description

Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.

You originally added the cells to the active list with either the [add_cell_models](#) command or with the *cell_type* library attribute. If you remove a cell model from the active list, you only remove the cell from that list and do not change the DFT library.

If you accidentally delete a DFT library cell from the active list with the [delete_cell_models](#) command, you can add the specified cell back into the active list with the [add_cell_models](#) command.

Arguments

- ***dftlib_model***

A repeatable string that specifies the names of the particular DFT library models that you want the tool to remove from the active list.

- **-Type INV | And | Buf | OR | NAnd | NOR | Xor | Mux | Scancell | DFF | LATCH**

A switch and argument pair that specifies the cell model type of all DFT library models that you want the tool to remove from the active list. The valid cell model types are as follows:

INV — A literal that specifies a one-input inverter gate.

And — A literal that specifies an AND gate.

Buf — A literal that specifies a one-input buffer gate.

OR — A literal that specifies an OR gate.

NAnd — A literal that specifies a NAND gate.

NOR — A literal that specifies a NOR gate.

Xor — A literal that specifies an exclusive OR gate.

Mux — A literal that specifies a 2-1 multiplexer.

Scancell — A literal that specifies a mux-scan D flip-flop.

DFF — A literal that specifies a D flip-flop.

LATCH — A literal that specifies a D latch.

- **-All**

A switch that removes all cell models from the active list, including those tagged in the DFT library with the *cell_type* attribute. This switch does not change the contents of the DFT library, only the active list within the tool.

Examples

The following example removes a DFT library model from the active list:

```
add_clocks 0 clk
set_test_logic -set on -reset on
set_system_mode analysis
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
delete_cell_models or2
insert_test_logic
```

Related Topics

[add_cell_models](#)

[report_cell_models](#)

[set_test_logic](#)

[delete_chain_masks](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup and analysis when EDT is off

Mode: setup only when EDT or LBIST is on

Removes chain masks from the specified scan chains.

Usage

```
delete_chain_masks chain_name... | -INStance instance_name -Block_chain_index_list  
          block_chain_index_list | -All
```

Description

Removes chain masks from the specified scan chains.

The delete_chain_masks command deletes chain masks from the specified scan chains defined with the [add_chain_masks](#) command. You can delete the definitions of specific chain name or all chains.

Arguments

- ***chain_name***

A repeatable string that specifies the names of the chain mask definitions that you want to delete.

- **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list*...**

Required set of switches and values that specifies a core instance object_spec and the list of chain indices starting with “1” on the EDT controller.

- **-All**

A switch that deletes all chain mask definitions.

Examples

The following example defines two chain masks then deletes all chain masks:

```
add_chain_masks chain2 chain3  
delete_chain_masks -all
```

Related Topics

[add_chain_masks](#)

[report_chain_masks](#)

delete_clocks

Context: all contexts

Mode: setup

Removes clocks from the clock list.

Usage

`delete_clocks pin_or_label_name... | -All`

Description

Removes clocks from the clock list.

The `delete_clocks` command removes clocks from the clock list that you can specify by pin name or label name. If you delete an equivalence pin, the command also deletes all of the equivalent pins from the clock list.

If you want to change one or more options of a defined clock, you can use the [set_clock_options](#) command to modify the clock.

Arguments

- *pin_or_label_name*
A repeatable string that specifies the pin name or label name to delete from the clock list.
- **-All**
A switch that deletes all clocks from the clock list.

Examples

The following example deletes an incorrectly added clock from the clock list:

```
add_clocks 1 clock1
add_clocks 1 clock2
delete_clocks clock1
```

Related Topics

[add_clocks](#)
[report_clocks](#)
[set_clock_options](#)

delete_config_element

Context: unspecified, all contexts

Mode: all modes

Deletes one or more configuration elements. Only elements which can be added can be deleted.

Usage

```
delete_config_element config_object | {config_name_list [-in_wrapper wrapper_object_spec]}
```

Description

Deletes one or more configuration elements. Only elements which can be added can be deleted.

Singular elements such as a wrapper declared with Repeatable set to single in the metadata and properties cannot be deleted as they always exist as soon as their parent wrapper exist.

Arguments

- *config_object*

A required collection containing one or more configuration elements. When using a collection of object, the -in_wrapper option cannot be used as the objects contains the full path of the object

- *config_name_list*

A required value that specifies a Tcl list containing names of one or more configuration elements. When the -in_wrapper option is specified, the names are relative to the specified wrapper element.

- -in_wrapper *wrapper_object_spec*

An optional value pair that is used to specify the parent wrapper in which the configuration elements are to be deleted. The *wrapper_object_spec* value is the name of a wrapper or a collection containing a wrapper element. The names in the *config_name_list* are relative to the specified *wrapper_object_spec*.

Examples

The following example deletes elements using the full name of elements, using a relative name with respect to a parent wrapper and using a collection of objects.

```
delete_config_element {
    DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(1)
    DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(2) }

delete_config_element {HostScanInterface(1) HostScanInterface(2)} \
    -in_wrapper DftSpecification(mod1,rtl)/IjtagNetwork

> delete_config_element [get_config_elements HostScanInterface \
    -in DftSpecification(mod1,rtl)/IjtagNetwork]
```

Related Topics

[add_config_element](#)

[get_config_elements](#)

[delete_config_messages](#)

Context: all contexts

Mode: all modes

Deletes error, warning or info message that were added to configuration elements using the add_config_message command.

Usage

```
delete_config_messages config_object_spec
    [-hierarchical] [-all | [-error] [-warning] [-info]] [-silent]
```

Description

Deletes error, warning or info message that were added to configuration elements using the add_config_message command.

Arguments

- ***config_object_spec***

A required value that specifies a configuration object on which to delete the messages. The config_object_spec value is the name of a configuration element or a collection containing a single configuration element object.

- **-hierarchical**

An optional Boolean option that specifies that the config message should be deleted on the specified config_object_spec and all elements below it. The -hierarchical option is ignored when the config_object_spec is a property.

- **-all**

An optional Boolean option that specifies that all message types are to be deleted. The use of the -all option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the four options is specified, -all is assumed.

- **-error**

An optional Boolean option that specifies that the messages to be deleted are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- **-warning**

An optional Boolean option that specifies that the messages to be deleted are of type warning. The use of the -error option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- **-info**

An optional Boolean option that specifies that the messages to be deleted are of type info. The use of the -error option is mutually exclusive to the specification of the -all option. When none of the -error, -warning and -info options is specified, -all is assumed.

- **-silent**

An optional Boolean option that disables the warning that is normally generated if the specified configuration element does not contain any messages of the specified type.

Examples

The following example deletes all message on and below the wrapper `DftSpecification(modal,rtl)`. The addition and deletion of messages on the `DftSpecification` and `DefaultsSpecification/DftSpecification` wrappers is automatically handled by the `process_dft_specification` command.

```
delete_config_messages DftSpecification(modal,rtl) -hierarchical
```

Related Topics

[add_config_message](#)
[report_config_messages](#)
[get_config_messages](#)

[delete_config_tabs](#)

Context: all contexts

Mode: all modes

Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using the [add_config_tab](#) or [display_specification](#) command.

Usage

```
delete_config_tabs [config_object_spec] [-silent]
```

Description

Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using the [add_config_tab](#) or [display_specification](#) command.

When no config_object_spec value is specified then all existing tabs are deleted.

Arguments

- *config_object_spec*
An optional value that specifies one or more configuration wrappers. The value is either the a tcl list of one or more names of wrappers or a collection containing one or more configuration wrappers. The wrappers are the root wrapper displayed in the tab.
- -silent
An optional value that disables the warning that is normally generated when *config_object_spec* contains a configuration element that is not a root wrapper displayed in an existing tab or when no configuration tab exists at all.

Examples

The first example deletes the tab in which the DftSpecification(mod1.rtl) wrapper is displayed. The second example deletes all tabs and suppresses the warning that would be generated if there are no tab to delete.

```
delete_config_tabs DftSpecification(mod1.rtl)
delete_config_tabs -silent
```

Related Topics

[add_config_tab](#)

[display_specification](#)

delete_connections

Context: all contexts

Mode: insertion

Disconnects net, pin, or port objects.

Usage

```
delete_connections obj_spec [-silent]
```

Description

Disconnects net, pin, or port objects.

If the disconnected object is a pin, the pin is left without a connection in the instance independent of whether the pin is an input, output, or inout pin. If the object is a port, if the associated net of the same name is connected to any other object, the tool creates a unqualified net and uses it in all of the connections the original net made. This results in input and inout ports with no fanout, and output and inout ports with no fanin. If the object is a net that is also a pin, the tool creates a unqualified net and uses it in all of the connections the original net made. This results in input and inout pins with no fanout, and output and inout ports with no fanin.

You must be careful when using the delete_connection command inside a non-unqualified design. You must only do the move or disconnection once per module and once per generate loop count. See how the parent_instance and the leaf_name_hash attributes are used in section 3 of Example 5 in the [get_dft_cell](#) command description to perform this robustly.

Arguments

- *obj_spec*

The obj_spec must be a Tcl list of one or more net, pin, or port names, or a collection of one or more net, pin or port names created by the get_nets, get_pins, or get_ports commands.

If an object within obj_spec does not exist and the -silent option is specified, the object is ignored. If an object within obj_spec does not exist and the -silent option is not specified, the tool generates an error.

- -silent

An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Examples

Example 1

The following example disconnects the net or constant that may be connected to input pin u1/A.

```
delete_connections u1/A
{ }
```

Example 2

The following example disconnects the net or constant that may be connected to the pin within the generated collection.

```
delete_connections [get_pins ABC*/TE]
{ }
```

Example 3

The following example disconnects the internal fanout of the input pin ABC/in1. It does that by renaming the net ABC/in1 to ABC/in1_ts1. The pin ABC/in1 ends up with no fanout.

```
delete_connections [get_nets ABC/in1]
{ABC/in1_ts1}
```

Example 4

The following example disconnects the internal fanin of the output port out1. It does that by renaming the net out1 to out1_ts1. The port out1 ends up with no fanin.

```
delete_connections out1
{out1_ts1}
```

Related Topics

[create_connections](#)
[get_nets](#)
[get_icl_pins](#)
[get_icl_ports](#)
[move_connections](#)

delete_core_descriptions

Context: all contexts

Mode: setup

Removes core descriptions that have been read in using the `read_core_descriptions` command from memory.

Usage

```
delete_core_descriptions {-all | -cores core_name_list} [-modes mode_name_list]
```

Description

Removes core descriptions that have been read in using the `read_core_descriptions` command from memory.

You cannot delete a core description that is instantiated in another core description unless you specify to delete the parent core description also, by using the `-all` switch for example.

If a corresponding core instance exists for the core description, this command deletes the core instance along with the core description.

If you specify to delete multiple descriptions and the deletion of one or more core descriptions fails, this command will continue deletion of the remaining core descriptions.

The deletion of a top-level core description is only allowed in a non-extraction flow.

Arguments

- **-all**
A switch that specifies to delete all core description files in the current design.
- **-cores *core_name_list***
A required switch and Tcl list that specifies the core names of the core descriptions to be deleted.
- **-modes *mode_name_list***
An optional switch that specifies the mode of the core to delete.

Examples

The following example reads in the core descriptions from the `top_core_des` file and then reports the defined core descriptions. The top-level core description is then deleted from memory. The `second_level_core` becomes the top of the design. The `report_core_descriptions` command then reports the defined core descriptions after the top-level core description is deleted.

```
set_context patterns -scan_retargeting
read_core_descriptions top_core_des
report_core_descriptions
```

Name	Mode Name	Mode Type
* top	retargeting	retargeting
second_level_core	retargeting	retargeting
core1	mode_M1	internal
	mode_M3	internal
core2	mode_M2	internal
core3	mode_M2	internal
	mode_M3	internal

```
delete_core_descriptions -core top
set_current_design second_level_core
report_core_descriptions
```

Name	Mode Name	Mode Type
* second_level_core	retargeting	retargeting
core1	mode_M1	internal
	mode_M3	internal
core2	mode_M2	internal
core3	mode_M2	internal
	mode_M3	internal

Related Topics

[add_core_instances](#)
[delete_core_instances](#)
[read_core_descriptions](#)
[report_core_instances](#)
[report_core_descriptions](#)
[write_core_description](#)

delete_core_instances

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Removes an instance binding created with the [add_core_instances](#) command.

Usage

```
delete_core_instances  
 {-all | -cores core_names | -modules module_objects | -instances instance_objects  
 | -current_design}  
 [-modes mode_names] [-silent]
```

Description

Removes an instance binding created with the [add_core_instances](#) command.

You can use this command to retarget patterns for different configurations sequentially in the same tool session. After completing one configuration and before switching to the next configuration, you use the `delete_core_instances` command to delete the old core instances so that the core instances used in the next session can be added.

Arguments

- **-all**
A required switch that specifies to remove all bindings that have been added with the [add_core_instances](#) command.
- **-cores *core_names***
A required switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be removed.
- **-modules *module_objects***
A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command removes the instance bindings for each instantiation of those modules in the design.
- **-instances *instance_objects***
A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command removes the instance bindings for each instance in the list.
- **-current_design**
A required switch that specifies to delete the core instance with respect to the current design. This switch is mutually exclusive with the -core, -modules, and -instances switches.

- **-modes mode_names**
An optional switch and string pair that specifies the list of modes to delete for each specified core instance.
- **-silent**
An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example adds the same instance twice. The second attempt fails with a warning, deletes it, and re-adds it successfully.

```
add_core_instances -instances core_1 -cores my_core
// Added core instance 'core_1'.

add_core_instances -instances core_1 -cores my_core
// Warning: Instance 'core_1' has already been added. Skipping.

delete_core_instances -instances core_1
add_core_instances -instances core_1 -cores my_core
// Added core instance 'core_1'.
```

Example 2

The following example adds two instances of the same module and then deletes both of them.

```
add_core_instances -instances {core_1 core_2} -cores my_core
// Added core instance 'core_1'.
// Added core instance 'core_2'.

delete_core_instances -instances {core_1 core_2}
```

Example 3

The following example adds two instances and then attempts to delete the two instances as well as one other instance that has not been added. The **-silent** usage suppresses the warning messages that would be printed for the three non-existing core instances.

```
add_core_instances -instances {core_1 core_2} -cores my_core
delete_core_instances -instances {core_1 core_2 core_3}
// Warning: Instance 'core_3' of module 'my_core_2' has not been added as
// a core instance. Skipping.

delete_core_instances -instances {core_1 core_2 core_3} -silent
```

Example 4

The following example adds two instances of the same module and then deletes both of them through the module.

```
add_core_instances -instances {core_1 core_2} -cores my_core
```

```
// Added core instance 'core_1'.  
// Added core instance 'core_2'.
```

```
delete_core_instances -modules my_module
```

Related Topics

[add_core_instances](#)

[read_core_descriptions](#)

[report_core_instances](#)

[write_core_description](#)

delete_design

Context: all contexts

Mode: setup

Removes all design modules including the flat model and all tool data that is design-specific.

Usage

delete_design

Description

Removes all design modules including the flat model and all tool data that is design-specific. Cell modules read in with the [read_cell_library](#) command are preserved. However, cell modules that are Verilog modules surrounded by `celldefined and loaded with the [read_verilog](#) command are also deleted.

This command also closes any open DFTVisualizer windows. Note that this command does not delete cell libraries unless a flat model is deleted.

When the **delete_design** command is deleting a flat model, and that flat model was saved with a list of enabled/disabled commands, then the restriction on the disabled commands is removed. In other words, all commands allowed in the tool will be enabled for the next design.

Arguments

None

Examples

The following example removes all design modules:

delete_design

Related Topics

[delete_cell_library](#)
[read_verilog](#)
[set_current_design](#)
[set_design_macros](#)
[read_flat_model](#)

delete_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Deletes a DFM rule from the open LDB.

Usage

```
delete_dfm [-rule rule_name] | [-vacuum]
```

Description

Deletes a DFM rule from the open LDB.

The -rule and -vacuum options are mutually exclusive.

Arguments

- **-rule *rule_name***

An optional switch that specifies a specific DFM rule to delete from the LDB. The rule name is one of the rule names as reported by the report_dfm_rules command. Enclose the rule name in quote marks.

When not specified, the tool deletes all the DFM rules from the open LDB.

- **-vacuum**

An optional switch that specifies to vacuum the LDB to reduce its size on disk. The tool recreates the LDB, moving records around to squeeze out any holes left after deleting DFM rules. For large databases, this operation may take a long time.

Examples

Example 1

The following example shows the DFM rules before and after deleting one rule from the LDB.

```
report_dfm_rules
delete_dfm -rule "metal_cross_edge_route1"
```

id	rule	layer	type
1	minimum_space_4_line_route1	route_1	BRIDGE
2	metal_cross_edge_route1	route_1	OPEN
3	low_density_min_space_route2	route_2	BRIDGE
4	half_comb_route3	route_2	OPEN
5	metal_cross_edge_route1	route_3	OPEN
6	corner_to_interior_metal4	route_4	BRIDGE
7	divergent_line_with_jog_metal5	route_5	BRIDGE
8	metal_cross_edge_route1	route_5	OPEN
9	route_5_barbell	route_5	OPEN
10	route_2_between_vias	route_2	OPEN

report_dfm_rules

id	rule	layer	type
1	minimum_space_4_line_route1	route_1	BRIDGE
3	low_density_min_space_route2	route_2	BRIDGE
4	half_comb_route3	route_2	OPEN
6	corner_to_interior_metal4	route_4	BRIDGE
7	divergent_line_with_jog_metal5	route_5	BRIDGE
9	route_5_barbell	route_5	OPEN
10	route_2_between_vias	route_2	OPEN

Example 2

The following example deletes multiple DFM rules. It shows you three ways you can specify deleting rules.

delete_dfm -rule "rule_name1 rule_name2"

or:

delete_dfm -rule "rule_name1" "rule_name2"

or:

delete_dfm -rule "rule_name1"

delete_dfm -rule "rule_name2"

Related Topics

[import_dfm](#)

[report_dfm_rules](#)

delete_dft_clock_enables

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the `set_current_design` command.

Deletes one or all previously added DFT clock enable signals.

Usage

```
delete_dft_clock_enables port_pin_spec | -all
```

Description

Deletes one or all previously added DFT clock enable signals.

All DFT clock enable signals are automatically deleted when elaborating a new design or re-elaborating the same design using the `set_current_design` command.

Arguments

- ***port_pin_spec***

A required string that specifies the name of a port or pin on the current design, or a collection containing a port or pin on the current design, or a port on a sub-module. The pin or ports must have previously been the target of an [add_dft_clock_enables](#) command.

- **-all**

A required Boolean switch that specifies that all pins and ports that were previously the target of an [add_dft_clock_enables](#) command are to be removed from the DFT clock enable signal list.

Examples

The following example adds three DFT clock enable signals, reports them, deletes one and reports them again, deletes them all and finishes with a last report. Notice that the clock enable signal defined on the port “FE” of the cell “cgand” remains even after deleting them all because this one was not defined with the `add_dft_clock_enables` command but instead was defined in the Tessent cell library.

```
add_dft_clock_enables en
add_dft_clock_enables blockb_i1/en2 -active_polarity 0
report_dft_clock_enables

// Dft clock enables
// =====
// -----  -----
//       Node      polarity
// -----  -----
// Port 'FE' of 'cgand'    1
// Port 'en'          1
// Pin   'blockb_i1/en2'  0
//
```

```
delete_dft_clock_enables en
report_dft_clock_enables

// Dft clock enables
// =====
// -----
//       Node      polarity
// -----
// Port 'FE' of 'cgand'    1
// Pin   'blockb_i1/en2'    0
//

delete_dft_clock_enables -all
report_dft_clock_enables

// Dft clock enables
// =====
// -----
//       Node      polarity
// -----
// Port 'FE' of 'cgand'    1
```

Related Topics

[add_dft_clock_enables](#)
[report_dft_clock_enables](#)
[read_cell_library](#)

delete_dft_clock_muxes

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the [set_current_design](#) command.

Deletes one or all previously added DFT clock multiplexers.

Usage

```
delete_dft_clock_muxes pin_port_spec | -all
```

Description

Deletes one or all previously added DFT clock multiplexers.

All DFT clock mux specifications are automatically deleted when elaborating a new design or re-elaborating the same design using the [set_current_design](#) command.

Arguments

- *port_pin_spec*

A required string that specifies the name of a port or pin on the current design, or a collection containing a port or pin on the current design. The pin or port must have previously been the target of an [add_dft_clock_mux](#) command.

- **-all**

A required Boolean switch that specifies that all pins and ports that were previously the target of an [add_dft_clock_mux](#) command are to be removed from the DFT clock mux list.

Examples

The following example adds three DFT clock multiplexers, reports them, deletes one and reports them, deletes them all and then reports them again.

```
add_dft_clock_muxes clk1 -test_clock_source mem1/clk -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_clock_muxes pll1/vco -test_clock_source pll1/ref \
    -dft_signal_source_name pll_bypass
add_dft_clock_muxes pll1/ref -test_clock_source clk1 -dft_signal_source_name alt_ref
report_dft_clock_muxes

// Dft clock muxes
// =====
//   Node      Test clock source  Control source name
//   -----  -----
//   'clk1'    mem1/clk          all_test
//   'pll1/vco'  pll1/ref        pll_bypass
//   'pll1/ref'  clk1            alt_ref
//
```

```
delete_dft_clock_muxes pll1/vco
report_dft_clock_muxes

// Dft clock muxes
// =====
// -----  -----
//   Node      Test clock source  Control source name
// -----  -----
// 'clkb'     mem1/clk          all_test
// 'pll1/ref' clkb             alt_ref
//



delete_dft_clock_muxes -all
report_dft_clock_muxes
// There are no defined dft clock muxes.
```

Related Topics

[add_dft_clock_mux](#)
[report_dft_clock_muxes](#)

delete_dft_control_points

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the [set_current_design](#) command.

Deletes one or all previously added DFT control points.

Usage

```
delete_dft_control_points pin_port_spec | -all
```

Description

Deletes one or all previously added DFT control points.

All DFT control point specifications are automatically deleted when elaborating a new design or re-elaborating the same design using the [set_current_design](#) command.

Arguments

- ***port_pin_spec***

A required string that specifies the name of a port or a pin on the current design or a collection containing a port or a pin on the current design. The pin or port must have previously been the target of an [add_dft_control_points](#) command.

- **-all**

An optional Boolean switch that specifies that all pins and ports that were previously the target of an [add_dft_control_points](#) command are to be removed from the DFT control point list.

Examples

The following example adds three DFT control points, reports them, deletes one and reports them, and then deletes them all and reports them again.

```
add_dft_control_points portb -dft_signal_source_name all_test  
register_static_dft_signal_names pll_bypass  
add_dft_control_points pll1_mux/s -dft_signal_source_name pll_bypass  
add_dft_control_points ref_mux/s -dft_signal_source_name alt_ref  
report_dft_clock_mux
```

```
// Dft clock muxes
// =====
// -----
//   Node      Control source name
// -----
// 'portb'    all_test
// 'pll1_mux/s' pll_bypass
// 'ref_mux/s' alt_ref
//

delete_dft_control_points pll1_mux/s

report_dft_control_points

// Dft clock muxes
// =====
// -----
//   Node      Control source name
// -----
// 'portb'    all_test
// 'ref_mux/s' alt_ref
//

delete_dft_control_points -all

report_dft_control_points

// There are no defined dft control points.
```

Related Topics

[add_dft_control_points](#)
[report_dft_control_points](#)

delete_dft_modal_connections

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the `set_current_design` command.

You use the `delete_dft_modal_connections` command to delete modal connection specifications previously added by the `add_dft_modal_connections` command.

Usage

```
delete_dft_modal_connections
{-ports port_spec | -auxiliary_data_pins pin_spec | -all}
```

Description

The `delete_dft_modal_connections` command is used to delete modal connection specifications previously added by the [add_dft_modal_connections](#) command. You can either delete all previously defined DFT modal connections using the `-all` switch or delete all connections to or from a set of ports or auxiliary data pins using the `-ports` or `-auxiliary_data_pins` switch. Use the `report_dft_modal_connections` command to see the list of ports or auxiliary data pins that can be referenced.

Arguments

- **`-ports port_spec`**

A switch used to specify a list of one or more port names or a collection containing one or more port objects. The ports must have been previously referenced using the “[add_dft_modal_connections -ports](#)” command.

- **`-auxiliary_data_pins pin_spec`**

A switch used to specify a list of one or more pin names or a collection containing one or more pin objects. The pins must have been previously referenced using the “[add_dft_modal_connections -auxiliary_data_pins](#)” command. An auxiliary data pin automatically inferred using the “[add_dft_modal_connections -auxiliary_data_pins](#)” command can also be referenced. The “Modes.” section of the report created with the `report_dft_modal_connections` command is useful to see all the port and auxiliary pins scheduled to be part of a modal connection.

- **`-all`**

A Boolean switch used to specify that all previously specified DFT modal connections are to be deleted.

Examples

The following example shows the result of the `report_dft_modal_connections` before and after a `delete_dft_modal_connections` command invocation. The starting configuration comes from [Example 1](#) of the [add_dft_modal_connections](#) command description.

SETUP> report_dft_modal_connections

```
Dft Modal Connections
=====
Modes
-----
edt_mode : DFT signal with usage scan_mode(unwrapped)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'

retargeting2_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connections (2):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
    Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'

Input connections
-----
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
    edt_mode : <Existing destination>
    retargeting1_mode : corea_i1/edt_channels_in[1]
    retargeting2_mode : corea_i2/edt_channels_in[1]
    retargeting3_mode : corea_i1/edt_channels_in[1]
                    : corea_i2/edt_channels_in[1]
2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
    edt_mode : <Existing destination>
    retargeting1_mode : corea_i1/edt_channels_in[0]
    retargeting2_mode : corea_i2/edt_channels_in[0]
    retargeting3_mode : corea_i1/edt_channels_in[0]
                    : corea_i2/edt_channels_in[0]
```

delete_dft_modal_connections

```

Output connections
-----
1) Port 'gpio[2]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_2_AuxOut'
    edt_mode : <Existing source>
    retargeting1_mode : corea_i1/edt_channels_out[0]
    retargeting2_mode : corea_i2/edt_channels_out[0]
    retargeting3_mode : corea_i1/edt_channels_out[0]

2) Port 'gpio[3]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_3_AuxOut'
    retargeting3_mode : corea_i2/edt_channels_out[0]

SETUP> delete_dft_modal_connections -ports gpio[3:2] SETUP>
report_dft_modal_connections

Dft Modal Connections
=====
Modes
-----
edt_mode : DFT signal with usage scan_mode(unwrapped)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'

retargeting2_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'

Input connections
-----
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
    edt_mode : <Existing destination>
    retargeting1_mode : corea_i1/edt_channels_in[1]
    retargeting2_mode : corea_i2/edt_channels_in[1]
    retargeting3_mode : corea_i1/edt_channels_in[1]
                    : corea_i2/edt_channels_in[1]

2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
    edt_mode : <Existing destination>
    retargeting1_mode : corea_i1/edt_channels_in[0]
    retargeting2_mode : corea_i2/edt_channels_in[0]
    retargeting3_mode : corea_i1/edt_channels_in[0]
                    : corea_i2/edt_channels_in[0]

```

[delete_dft_signals](#)

Context: dft

Mode: setup

Deletes a DFT signal that was previously added using the [add_dft_signals](#) command.

Usage

```
delete_dft_signals {name ...} | -all_user_added
```

Description

A command used to delete a DFT signal that was previously added using the [add_dft_signals](#) command. Only DFT signals that were added and not yet created can be deleted. Once a DFT signal is inserted using the [process_dft_specification](#) command, the signal exists and can no longer be deleted.

Arguments

- ***name***

A repeatable string defining one or more DFT signal names to delete. You can specify several names in one command line invocation. You can also supply several names in a single well formatted Tcl list.

- **-all_user_added**

A Boolean switch that specifies the deletion of all DFT signals that were previously added using the [add_dft_signals](#) command.

Examples

The following example shows how to delete one DFT signal at a time or all of them at one time:

```
add_dft_signals ltest_en int_ltest_en ext_ltest_en  
get_dft_signals -list  
ltest_en int_ltest_en ext_ltest_en  
delete_dft_signals ltest_en  
get_dft_signals -list  
int_ltest_en ext_ltest_en  
delete_dft_signals -all_user_added  
get_dft_signals -list
```

Related Topics

[add_dft_signals](#)

[get_dft_signal](#)

delete_dft_signals

[process_dft_specification](#)

[report_dft_signals](#)

delete_display_callout

Context: all contexts

Mode: setup, analysis

Deletes a callout box from the schematic window.

Usage

```
delete_display_callout object_spec [-DIsplay {FLat_schematic| HIEarchical_schematic }]
```

Description

Deletes a callout box from a specified object, or list of objects, that are displayed in the Flat Schematic or Hierarchical Schematic window. The object associated with the callout box will not be removed. The [delete_display_instances](#) command should be used if it is desired to remove the object.

Arguments

- ***object_spec***

The value of *object_spec* can be a Tcl list or Tcl collection of one or more names of port, pin, net, pseudo_port, instance or gate_pin objects.

- **-DIsplay {FLat_schematic| HIEarchical_schematic }**

An optional switch and literal pair that specifies the schematic window from which the tool should delete the callout box of the specified object. By default, the object is assumed to be in the Flat Schematic window if the -display argument is not provided.

Examples

The following example deletes the callout box for the D input of instance reg_d_1_0_ that is displayed in the Hierarchical Schematic window:

```
delete_display_callout reg_d_1_0_/D -display hierarchical_schematic
```

Related Topics

[add_display_callout](#)

delete_display_data

Context: all contexts

Mode: setup, analysis

Removes a data column from the DFTVisualizer Data window.

Usage

```
delete_display_data
  Error_pattern |
  {PATtern_index pattern_index [-Internal] | -External]} |
  {Parallel_pattern pattern_number} |
  Fault_status |
  TlE_value |
  Constrain_value |
  Seq_depth_data |
  {Clock_cone pin_name} |
  CAPTURE PROCedure procedure_name |
  Drc_pattern {{Test_setup [{-{Cycle | -Time} n1} [n2 | End]]}} | Load_unload | SHIft |
    SKew_load | SHADOW_Control | Master_observe |
    SHADOW_Observable | STate_stability | TEST_End [time | -All]}
```

Note

 The following arguments are not supported in dft -scan: Pattern_index, Parallel_pattern, Fault_status, and Seq_depth_data.

Description

Removes a data column from the DFTVisualizer Data window.

You can control the display of a column by right clicking on the title bar and selecting/deselecting the appropriate option.

Arguments

This command has the same arguments and usage as the [add_display_data](#) command.

Examples

The following example adds a column named “drc test_setup -cycle 5 10” that displays the simulated values for cycles 5 through 10 of the test_setup procedure:

```
add_display_data drc test_setup -cycle 5 10
```

The following example deletes the column added in the preceding example:

```
delete_display_data drc test_setup -cycle 5 10
```

Note

 The argument string you specify when deleting a column must match exactly the column name, including spaces.

Related Topics

[add_display_data](#)

delete_display_instances

Context: all contexts

Mode: setup, analysis

Prerequisites: DFTVisualizer must be open and displaying instances.

Removes the specified design instances from a DFTVisualizer display window.

Usage

```
delete_display_instances {object_spec | -All} [-Display window_name]
```

Description

Removes the specified design instances from a DFTVisualizer display window.

Arguments

- *object_spec*

The value of *object_spec* can be the name of a port, pin, net, pseudo_port or instance object, a gate_id, or the name or id of a gate_pin. The value of *object_spec* can also be a Tcl list or Tcl collection of one or more names of port, pin, net, pseudo_port, instance or gate_pin objects.

- **-All**

A switch that removes all the instances in the specified DFTVisualizer display.

- **-Display *window_name***

An optional switch and literal that specifies which DFTVisualizer windows to delete instance data from. You can specify multiple windows. Literal options include:

FLAt_schematic — A literal that specifies the Flat Schematic window.

HIEarchical_schematic — A literal that specifies the Hierarchical Schematic window.

DAta — A literal that specifies the Data window.

WAve — A literal that specifies the Wave window.

The default is to delete instances from the Flat Schematic window only.

Related Topics

[add_display_data](#)

[add_display_instances](#)

[open_visualizer](#)

[report_display_instances](#)

delete_display_path

Context: all contexts

Mode: setup, analysis

Deletes specified objects or paths in the schematic window.

Usage

```
delete_display_path object_spec
  {-direction {forward | backward}}
  [-DIsplay {FLat_schematic | HIEarchical_schematic}]
```

Description

Removes single or multiple objects that are displayed in the specified or default schematic window.

Arguments

- ***object_spec***

The value of *object_spec* can be a Tcl list or Tcl collection of one or more names of pin, port, pseudo_port or gate_pin objects.

If *object_spec* is a single object, the object is removed from the schematic along with its associated instance if it is the last remaining pin. An error message is generated if the specified object does not exist, or is not of type pin, port, pseudo_port or gate_pin.

The *object_spec* list or collection may also contain lists of names of pins, ports or pseudo ports that form a contiguous path in the schematic in the direction specified by the direction switch.. Each pair is handled as a segment of the path to be removed, where one of the objects is a fan-in of the other. All pairs should have the same direction of fan-in and fan-out, otherwise an error is generated. An error is also generated if an object does not exist or there is no valid path between the pair.

- **-direction {forward | backward}**

A required switch and literal pair that specifies the direction of the path provided in *object_spec*.

- **-DIsplay {FLat_schematic | HIEarchical_schematic}**

An optional switch and literal pair that specifies the schematic window to which the tool should remove the specified object or path. By default, the object or path will be removed from the Flat Schematic window if the -display argument is not provided.

Examples

The following example removes the pin object ix172/A that is currently displayed in the Hierarchical Schematic window:

```
delete_display_path ix172/A -direction forward -display hierarchical_schematic
```

delete_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes the data for the specified EDT block(s) from the tool's internal database.

Usage

`delete_edt_blocks -All | block_name`

Description

Removes the data for the specified EDT block(s) from the tool's internal database.

The `delete_edt_blocks` command removes all data for the specified EDT block(s) from the tool's internal database, making the block(s) unavailable for use with the [set_current_edt_block](#) command.

If you delete the current EDT block, leaving just one remaining EDT block defined, the tool automatically sets the EDT context to the remaining block. The tool does not reset the EDT context, however, if there is more than one defined EDT block remaining; in this case subsequent context-sensitive commands that define attributes for a given EDT block will return an error until you set the EDT context using the [add_edt_blocks](#) or [set_current_edt_block](#) command.

You set or change the EDT context when you restrict the applicability of certain commands to a particular EDT block by issuing the `add_edt_blocks` or [set_current_edt_block](#) command. The EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the `delete_edt_blocks` command or change the context to a different EDT block using the `add_edt_blocks` or [set_current_edt_block](#) command. The `add_scan_chains` command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. Most commands like `set_system_mode`, `report_statistics` and `create_patterns`, are unaffected by the EDT context and always operate on the whole design.

Arguments

- **-All**

A switch that specifies to remove the data for all EDT blocks from the tool's internal database.

- ***block_name***

A string that specifies to remove the data for a particular EDT block from the tool's internal database.

Examples

Example 1

The following example displays currently defined EDT block names, then removes the definition for the one representing the current EDT block.

```
report_edt_blocks
// my_core1_block
// my_core3_block      (current block)

delete_edt_blocks my_core3_block
report_edt_blocks

// Block deleted. my_core1_block is now the current EDT block.
```

Example 2

The following example shows how the tool manages the current EDT context when there are more than two blocks defined.

```
report_edt_blocks
// my_core1_block
// my_core2_block
// my_core3_block
// my_core4_block      (current block)

delete_edt_blocks my_core3_block
report_edt_blocks

// my_core1_block
// my_core2_block
// my_core4_block      (current block)

delete_edt_blocks my_core4_block

// Block deleted. No EDT block now selected as current block.

set_edt_options -channels 4

// Error: No current EDT block defined.

set_current_edt_block my_core2_block
set_edt_options -channels 4
report_edt_configuration
```

```
//  
// EDT block "my_core2_block"  
// -----  
// IP version: 3  
// External scan channels: 4  
// Bypass logic: On  
// Lockup cells: On  
// Clocking: edge-sensitive  
//  
// Note: Only current EDT block "my_core2_block" is reported. Use "report  
// edt configuration -all_blocks" to report the configurations of all  
// blocks.
```

Related Topics

[add_edt_blocks](#)

[report_edt_blocks](#)

[set_current_edt_block](#)

delete_edt_configurations

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the [add_edt_configurations](#) command.

Deletes specified compression configurations from all blocks in a design.

Usage

`delete_edt_configurations -All | config_name`

Description

Deletes specified compression configurations from all blocks in a design.

In a multi-block design flow, all subsequently added blocks inherit the defined compression configurations. The compression configuration parameters are then set for each block context.

Arguments

- **-All**

A required switch that deletes all the compression configurations from all the blocks of the design.

- ***config_name***

A required string that specifies the name of a single compression configuration to delete from all the blocks of the design.

Examples

The following example reports the currently defined compression configurations in a design, deletes one, and reports the remaining compression configurations.

report_edt_configuration

```
// my_core1_block
// IP version:                      1
// Bypass logic:                    On
// Lockup cells:                   On
// Clocking:                        edge-sensitive
// Input channels:      [manufacturing_test : 10] [system_test : 4]
// Output channels:     [manufacturing_test : 5] [system_test : 2]
// Bypass chains:      5
```

delete_edt_configurations system_test
report_edt_configuration

```
// my_core1_block
// IP version: 1
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10]
// Output channels: [manufacturing_test : 5]
// Bypass chains: 5
```

Related Topics

[add_edt_configurations](#)
[report_edt_configurations](#)
[set_current_edt_configuration](#)

delete_false_paths

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes definitions of false paths.

Usage

```
delete_false_paths [-All] [-Verbose]
  [-From source_node... | -FROM_Clock clock_signal_name...]
  [-TO sink_node... | -TO_Clock clock_signal_name...]
  [{-THrough through_node...}...]
```

Description

Removes definitions of false paths.

The delete_false_paths command deletes from the tool's internal database, false paths you defined using the [add_false_paths](#) command.

Arguments

Tip When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- **-All**
An optional switch that specifies to delete all previously defined false paths.
- **-Verbose**
An optional switch that directs the tool to display the definition of each deleted false path as it is deleted.
- **-From *source_node...***
An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.
If a specified source node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- **-FROM_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- **-TO *sink_node*...**

An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

If a specified sink node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

Note

 For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “`delete_false_paths... -to CLK2`” would delete false paths that have an end point at D2.

- **-TO_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- **{-THrough *through_node*...}...**

An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified false path. For example, the following false path specification:

```
delete_false_paths -from A -through B C -through D E -to F
```

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool will consider all paths from the specified starting points to the specified end points to be false paths.

If a through node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

Examples

The following example displays current false path definitions, then deletes one of them, echoing its definition as it is deleted:

```
report_false_paths -all
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3

delete_false_paths -verbose -to /my_design/c

// Deleting false paths ...
// False Path -to /my_design/c ( /my_design/c/Y (36) )
// Deleted 1 false paths.
```

Refer to the [add_false_paths](#) command for examples of signal matches for pathnames specified using the asterisk (*) character.

Related Topics

[add_false_paths](#)
[delete_multicycle_paths](#)
[read_sdc](#)
[report_false_paths](#)
[report_multicycle_paths](#)

delete_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.

Usage

Bridge Faults Usage

```
delete_fault_sites {-All | {bridge_name... -NAME}}
```

Path Delay Faults Usage

```
delete_fault_sites {-All| -Unsensitizable_paths | path_name...}
```

UDFM Faults Usage

```
delete_fault_sites {-All | file_name}
```

Description

Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.

Depending on the current fault model, deletes specified fault sites (bridge entries or delay paths) from the current fault list. The `delete_fault_sites` command removes faults loaded with the `read_fault_sites` command. When a fault site is removed from the current fault list, the associated patterns are also removed from the internal pattern set.

For more information on bridge faults and the bridge fault definition file, refer to “[The Static Bridge Fault Model](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information on path delay faults and the path definition file, refer to “[Path Delay Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information on UDFMs, see “[About User-Defined Fault Modeling](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **-All**

A switch that specifies to remove all fault sites from the current fault list and deletes all faults associated with the deleted fault sites.

- **bridge_name**

A repeatable string that specifies a bridge name.

- **-NAME**

A switch that specifies the argument strings preceding the switch are bridge names. This switch is required just once, whether you enter one or multiple *bridge_name* strings.

- **-Unsensitizable_paths**

A switch that directs the tool to perform an ATPG analysis of each path, and to delete those proved unsensitizable. For a list of critical paths identified by static timing analysis, for example, use this switch to identify and delete the unsensitizable paths.

Note

 This ATPG process may be very lengthy. A progress message is displayed after every 100 paths analyzed, as well as at the end of the analysis.

- ***path_name***

A repeatable string that specifies the name of an existing path that resides in the current path definition file and whose path delay faults you want to remove from the current path delay fault list.

- ***file_name***

A required string that specifies a UDFM file name. Use this option to delete all fault sites associated with a specified UDFM file.

Examples

Example 1

The following example reads the bridge information in the bridge fault definition file, */user/design/bridgefile*, then deletes two of the three valid entries added to the fault site list:

```
set_fault_type bridge
read_fault_sites /user/design/bridgefile

//
// Load bridge entries from file "/user/design/bridgefile".
//
// 5 bridge entries were read from file "/user/design/bridgefile".
// 2 bridge entries were skipped due to unknown net name.
// 3 bridge entries were added to the fault site list.

report_fault_sites
```

delete_fault_sites

```

BRIDGE "U7.BRIDGE" {
    NET1 = "/G5";
    NET2 = "/G10";
    FAULTS = {"-", "-", "-", "-", "-"};
}
BRIDGE "U5.BRIDGE" {
    NET1 = "/G4";
    NET2 = "/G10";
    FAULTS = {"-", "-", "-", "-", "-"};
}
BRIDGE "U0.BRIDGE" {
    NET1 = "/G11";
    NET2 = "/G10";
    FAULTS = {"-", "-", "-", "-", "-"};
}

```

**delete_fault_sites U5.BRIDGE U7.BRIDGE -name
report_fault_sites**

```

BRIDGE "U0.BRIDGE" {
    NET1 = "/G11";
    NET2 = "/G10";
    FAULTS = {"-", "-", "-", "-", "-"};
}

```

Example 2

The following path delay example reads the path information from the file, */user/design/pathfile*, and deletes one of the two paths:

```

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```

```

PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
PATH "path1" =
    PIN /I$6/Q + ;
    PIN /I$35/B1 + ;
    PIN /I$35/C1 + ;
    PIN /I$1/I$649/IN + ;
    PIN /I$1/I$649/OUT - ;
    PIN /I$5/D - ;
END ;

```

delete_fault_sites path0
report_fault_sites

```
PATH "path1" =
PIN /I$6/Q + ;
PIN /I$35/B1 + ;
PIN /I$35/C1 + ;
PIN /I$1/I$649/IN + ;
PIN /I$1/I$649/OUT - ;
PIN /I$5/D - ;
END ;
```

Example 3

The following example deletes all fault sites associated with the *c090_std.udfm* UDFM model:

```
delete_fault_sites /home/design/c090_std/c090_std.udfm
```

Example 4

The following example deletes all fault sites from the current fault list:

```
delete_fault_sites -All
```

Related Topics

[add_fault_sites](#)

[add_faults](#)

[analyze_fault](#)

[delete_faults](#)

[read_fault_sites](#)

[report_fault_sites](#)

[set_fault_type](#)

[write_fault_sites](#)

delete_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Removes faults from the current fault list.

Usage

```
delete_faults fault_model_specific_options [power_aware_options] [-KEEP_Patterns] [{> | >>} file.pathname]
```

Path Delay Faults Usage

```
delete_faults {-All | name_of_delay_path...} [-Class {class_name | class_code}...] [-Both | -Rise | -Fall] [-UNTestable]
```

Stuck/Transition/Toggle/Iddq Faults Usage

```
delete_faults { { { {-All |  
-Class {class_name | class_code}...} |  
{object_expression... [-PIN | -INstance | -MODule [-PIN module_pin_pathname ...]]} |  
{-CLOCK_Domains {ALL | clock_pathname...} [-NO_EQUIvalent_clocks]  
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]} |  
{-CAPture_procedures {ALL | capture_procedure_name...}}}  
[-FRom pin...] [-THrough pin...] [-TO pin...]  
[-STOP_at {sequential_elements | scan_cells | ports_only}] [-Stuck_at [01 | 0 | 1]]} |  
{[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]} } [-VERbose]
```

Bridge Faults Usage

```
delete_faults {-All | {-Class {class_name | class_code}...}  
| {{bridge_name... -NAME} | {net_pathname... -SINGle}  
| {instance_name... -SINGLE}}  
| {-CLOCK_Domains {ALL | clock_pathname...} [-NO_EQUIvalent_clocks]  
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}  
| {-CAPture_procedures {ALL | capture_proc_name...}}  
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}}
```

UDFM Fault Usage

```
delete_faults {-All | {-Class {class_name | class_code} ...}  
| {object_expression ... [-INstance]}  
| {-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUIvalent_clocks]  
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}  
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}  
| {[-UDFM_type name] [-CELLname] [-MODULE name]  
[-FAULT name] } [-FRom pin...] [-THrough pin...] [-TO pin...]  
[-STOP_at {sequential_elements | scan_cells | ports_only}]  
[-VERBose]}
```

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

```
delete_faults [[-ON_domains] | [-OFF_domains] | [-POWER_domains {domain_name ...}]]  
[-ISolation_cells] [-LLevel_shifters] [-REtention_cells]
```

Description

Removes faults from the current fault list.

The delete_faults command deletes from the fault list, faults you added using the add_faults or read_faults command. The power-aware options remove faults based on power domains and/or power features after you have loaded a CPF or UPF file.

Alternatively, you can also delete a large number of faults using the [read_faults](#) command's -delete switch.

When you issue this command, the tool discards all patterns in the current test pattern set. To save the current test patterns you must explicitly save them with the [write_patterns](#) command prior to issuing the delete_faults command.

By default, the delete_faults command ignores the constraint values implied in the data path when deleting faults in the data path. However, the tool still considers the constraint values for clock path faults.

You cannot use a single command to delete_faults for clock domains, capture procedures, and specific classes, objects, or paths. Use a separate command instance for each.

When bridge faults are deleted, the currently loaded faults sites remain intact.

Arguments

- **-KEEP_Patterns**

An optional switch that specifies that the tool preserves the current test patterns while deleting faults. The default behavior is for the tool to delete the current test patterns as faults are deleted.

- **-All**

A required switch that deletes all faults in the current fault list.

- **name_of_delay_path**

A required, repeatable string that specifies the name of the delay path to delete from the current fault list. The name must correspond to the name of a delay path defined in [the path definition file](#).

- **-Class {class_name | class_code}...**

A required switch and repeatable literal pair that deletes one or more classes of faults. The table below lists the valid fault *class_code* and their associated fault *class_names*.

Table 3-15. Fault Class Codes and Class Names

Fault Class Codes	Fault Class Names	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- -Both | -Rise | -Fall

An optional switch that specifies which faults to delete for each path already added via the [read_fault_sites](#) command. These switches are valid only for path delay faults.

- Both — Deletes both the slow-to-rise and slow-to-fall faults. This is the default.
- Rise — Deletes only the slow-to-rise faults.
- Fall — Deletes only the slow-to-fall faults.

- *object_expression*

A required string representing a list of instances or pins associated with the faults you want to delete from the current fault list. Note that in the case of UDFM, you can specify only instances. You can use regular expressions that include any number of asterisk (*) and question mark (?) wildcard characters.

Instance pathnames must be Tessent Cell library cell instances. Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, it is ignored. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- **-PIN**

An optional switch that uses the *object_expression* to match only pin pathnames and then, delete the associated faults.

- **-INstance**

An optional switch that uses the *object_expression* to match only instances and then, delete the associated faults.

- **-MODule [-PIN *module_pin_pathname* ...]** (Stuck/Transition/Toggle/Iddq only)

A switch that specifies to interpret the *object_expression* argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the *object_expression* argument, and the tool adds the fault for all matching modules or library models.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and deletes faults for every pin specified in module_pin_pathname list.

- **-Untestable**

An optional switch that deletes all identified, untestable path delay faults. Untestable faults are common when using random patterns. This includes faults that the tool cannot detect due either to constraints or the use of a single capture clock.

- ***bridge_name***

A required, repeatable string that specifies the name of a bridge. The faults associated with the specified bridge will be deleted from the current fault list.

- ***net_pathname***

A required, repeatable string that specifies the name of a single net. The faults associated with the specified net will be deleted from the current fault list.

- ***instance_name***

A required, repeatable string that specifies the name of an instance. The faults associated with the specified instance will be deleted from the current fault list.

- **-NAME**

An optional switch that specifies all entered strings are bridge names.

- **-SINGle**

Optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

If the string specifies a net, each bridge entry associated with the net is deleted.

If the string specifies an instance that contains a non-dominant net included in a bridge entry, the associated faults are deleted.

- **-CLOCK_Domains {ALL | *clock_pathname...*}**

A required switch and literal or repeatable string pair that specifies a list of clocks which is used by the tool to decide the faults to be added to the fault list, given the following requirements are met:

- **Static Faults** — A fault is added to the fault list if it can be captured by any clock in the specified list of clocks or any of its equivalent clocks.
- **Transition Faults** — A fault is added to the fault list:
 - i. If the launch and capture clock are the same clock from the specified list of clocks
 - ii. If the launch and capture clock are synchronous clocks from the specified list of clocks
 - iii. If the launch or capture clock from the specified are equivalent equivalent clocks

When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

ALL — A literal that specifies all the clocks in the design.

clock_pathname — A repeatable string that specifies a particular clock.

Be aware that a fault potentially detectable by a clock you specify with this switch will be deleted even if it might also be detectable by an unspecified clock.

The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when adding or deleting faults by clock domain.

- **-NO_EQUIvalent_clocks**

An optional switch that prevents the -Clock_domains switch from deleting faults in equivalent clock domains.

- **-EXCLUDEFAULTSBETWEENSYNCclockdomains**

An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When specified, only faults within clock domains are considered when adding or deleting faults by clock domain.

- **-CAPture_procedures {ALL | *capture_procedure_name...*}**

A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to delete_faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

ALL — A literal that specifies all enabled named capture procedures.

capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

See the [set_capture_procedures](#) command description for information about enabling or disabling named capture procedures.

- **-FFrom pin...**

This option sets the start pin(s) of the cone of logic used to select faults to delete. It directs the tool to perform forward cone tracing from the specified pin(s) until it encounters a stop point. You can specify one or more instance names, rather than pin names. When you specify an instance name with -from, its output pins are used when defining the cone.

- **-THrough pin...**

This option sets the through pin(s) of the cone of logic used to select faults to delete. When specified, the tool performs forward and backward cone tracing from the specified pin(s) until it encounters a stop point in both directions. You can specify one or more instance names, rather than pin names. When you specify an instance name with -through, its output pins are used when defining the cone.

- **-TO pin...**

This option sets the end pin(s) of the cone of logic used to select faults to delete. When specified, the tool performs backward cone tracing from the specified pin(s) until it encounters a stop point. You can specify one or more instance names, rather than pin names. When you specify an instance name with -to, its input pins are used when defining the cone.

Note

 When -from, -to, and -through are used with other switches, the selected faults will be the intersection of the cones traced for all specified switches.

- **-STOP_at {sequential_elements | scan_cells | ports_only}**

This option sets the stop condition for tracing of the cone of logic used to select faults to delete. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify “scan_cells”, the cone includes non-scan cells and tracing stops only at scan cells. When you specify “ports_only”, the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

- **-Stuck_at 01 | 1 | 0**

An optional switch and literal pair that specifies the stuck-at or transition faults to delete from the fault list.

01 — A literal specifying that for stuck-at faults the tool delete both stuck-at-0 and stuck-at-1 faults; or for transition faults the tool delete both slow-to-rise and slow-to-fall faults. This is the default.

0 — A literal specifying to delete only the stuck-at-0 faults (slow-to-rise faults for transition faults).

1 — A literal specifying to delete only the stuck-at-1 faults (slow-to-fall faults for transition faults).

The following switches, which support stuck-at, transition, toggle, Iddq, UDFM, and bridge faults, delete faults at certain locations that are ATPG untestable in functional mode or should not be tested (to avoid over-testing). For bridge faults, the switches apply to any bridge that has at least one end in a specified location. When justified by your knowledge of the design, use the switches to fine-tune the fault population and improve the quality of the tool's coverage statistics.

- **-SCAN_Enable**

An optional switch that deletes faults that only fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch deletes all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- **-CLOCK_Cones**

An optional switch that deletes all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fan-out of the clock sources. This switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.

- **-IO**

An optional switch that deletes faults that are either only controlled by PIs or only observed by POs. This switch only applies to PIs that are either not defined as a clock or defined as clock (or read/write control), but constrained to be off during capture.

- **-ASYnchronous_controls**

An optional switch that deletes all faults that only fan out to Set/Reset ports of state elements and RAMs.

- **-VERbose**

An optional switch that outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs all found instances or pins matching the wildcard characters. For example:

```
ANALYSIS> delete_faults */irep0/sigdout -verbose
== Found 1 design pin ==
path_to_instance/irep0/sigdout
```

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- **>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

- **-Udfm_Type name**

A required switch and string pair that specifies a type of UDFM fault to delete from the current fault list. The name can contain wildcards.

- **-CELL name**

A required switch and string pair that specifies a library cell. Use this option to delete all faults associated with a specified cell. The name can contain wildcards.

- **-MODULE name (UDFM only)**

A required switch and string pair that specifies a module. Use this option to delete all faults associated with a specified module. The name can contain wildcards.

- **-Class class_type**

A required switch and repeatable literal pair that deletes the internal fault entries that belong to the class specified by the *class_type* argument. The *class_type* argument can be either the fault class code or a fault class name. For more information on fault classes, see “[Fault Classes](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-FAult name**

A required switch and string pair that specifies a UDFM fault name. The name can contain wildcards.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- **-ON_domains**

An optional switch that removes faults on all power-on domains.

- **-OFF_domains**

An optional switch that removes faults on all power-off domains.

- **-Power_domains {domain_name ...}**

An optional switch and repeatable string that removes faults on the specified power domains (*domain_name*).

- **-Isolation_cells**

An optional switch that removes faults on all isolation cells.

-
- **-Level_shifters**
An optional switch that removes faults on all level shifters.
 - **-REtention_cells**
An optional switch that removes faults on all retention cells.

Examples

Example 1

The following example deletes a stuck-at-0 fault from the current fault list after adding all the faults to the circuit, but before creating patterns:

```
set_system_mode analysis
add_faults -all
delete_faults i_1006/i1 -stuck_at 0
create_patterns
```

Example 2

The following example deletes all stuck-at-0 AU faults. Notice the reduction in AU faults in the statistics report and the corresponding increase in the reported coverages as a result of removing these faults from the calculation. (For information on the formulas used to calculate coverage, refer to “[Testability Calculations](#)” in the *Tessent Scan and ATPG User’s Manual*.)

```
ATPG> report_statistics
```

```

Statistics Report
Stuck-at Faults
-----
Fault Classes          #faults
                        (total)
-----
FU (full)           136996
-----
UO (unobserved)      84 ( 0.06%)
DS (det_simulation) 112635 (82.22%)
DI (det_implication) 10037 ( 7.33%)
PU (posdet_untestable) 199 ( 0.15%)
PT (posdet_testable) 7 ( 0.01%)
UU (unused)          62 ( 0.05%)
TI (tied)            96 ( 0.07%)
BL (blocked)         1 ( 0.00%)
RE (redundant)       9867 ( 7.20%)
AU (atpg_untestable) 4008 ( 2.93%)
-----
Fault Sub-classes
-----
AU (atpg_untestable)
    BB (black_boxes)      2278 ( 1.66%)
    Unclassified          1730 ( 1.26%)
UC+UO
    AAB (atpg_abort)     84 ( 0.06%)
-----
Coverage
-----
test_coverage        96.70%
fault_coverage        89.62%
atpg_effectiveness   99.94%
...

```

ATPG> delete_faults -class au -stuck_at 0

// Note: 619 collapsed faults were deleted.

ATPG> report_statistics

```

Statistics Report
Stuck-at Faults
-----
Fault Classes          #faults
                        (total)
-----
FU (full)             135789
-----
UO (unobserved)      84 ( 0.06%)
DS (det_simulation)  112635 (82.95%)
DI (det_implication) 10037 ( 7.39%)
PU (posdet_untestable) 199 ( 0.15%)
PT (posdet_testable)  7 ( 0.01%)
UU (unused)           62 ( 0.05%)
TI (tied)              96 ( 0.07%)
BL (blocked)           1 ( 0.00%)
RE (redundant)         9867 ( 7.27%)
AU (atpg_untestable) 2801 ( 2.06%)
-----
Fault Sub-classes
-----
AU (atpg_untestable)
BB (black_boxes)       1654 ( 1.22%)
Unclassified           1147 ( 0.84%)
UC+UO
AAB (atpg_abort)      84 ( 0.06%)
-----
Coverage
-----
test_coverage        97.62%
fault_coverage       90.42%
atpg_effectiveness   99.94%
...

```

Example 3

After adding all of the faults in the design, this example removes only the faults from the power domain named PD1:

```
add_faults -all
delete_faults -power_domain PD1
```

Example 4

The following example deletes all UDFM type *intra_cell_bridges* associated with all library cells that begin with *mux*:

```
delete_faults -udfm_type intra_cell_bridges -cell mux*
```

Related Topics

[add_faults](#)
[delete_nofaults](#)
[read_fault_sites](#)
[read_faults](#)

[report_fault_sites](#)
[report_faults](#)
[report_testbench_simulation_options](#)
[set_fault_mode](#)
[set_fault_sampling](#)
[set_fault_type](#)
[write_fault_sites](#)
[write_faults](#)
[write_patterns](#)

delete_flat_model

Context: all contexts

Mode: setup

Deletes a flat model that was previously created either by going to system mode analysis or by using the [create_flat_model](#) command.

Usage

```
delete_flat_model
```

Description

Deletes a flat model that was previously created either by going to system mode analysis or by using the [create_flat_model](#) command.

You typically do not need this command because any command you use that invalidates the flat model automatically deletes the flat model at the same time. The tool also automatically deletes the flat model when you enter insertion mode.

In addition, the tool deletes the flat model when you set an attribute on a design module, instance, port, or pin that has the preserve boundary option enabled. For more information, refer to the “[set_attribute_options -preserve_boundary_in_flat_model](#)” description.

To free up used memory, you can use this command to delete the flat model while still in setup mode if you do not need it anymore.

You cannot use this command to delete a flat model that was read in with the [read_flat_model](#) command. In this case, you must use the [delete_design](#) command to delete the flat model.

Arguments

None

Return Values

None

Related Topics

[create_flat_model](#)

[delete_design](#)

[read_flat_model](#)

[set_attribute_options](#)

delete_icl_modules

Context: all contexts

Mode: setup

Deletes the specified ICL modules from memory.

Usage

```
delete_icl_modules {object_spec | -all} [-silent]
```

Description

Deletes the specified ICL modules from memory.

If any of the specified ICL modules are instantiated below the current design, the ICL elaboration tree is destroyed and you must recreate it using a new [set_current_design](#) command.

Arguments

- *object_spec*

A value that specifies one or more ICL modules to delete. The value is a Tcl list of one or more ICL module names, or a collection of one or more ICL module objects as returned by the [get_icl_modules](#) command. When this argument is omitted, all ICL modules are deleted.

- **-all**

A switch that specifies to delete all ICL modules in the system.

- **-silent**

An optional argument that specifies to not generate error messages when an ICL module specified by *object_spec* does not exist.

Return Values

None

Examples

The first example deletes three ICL modules. The second example deletes all ICL modules. The third example deletes all ICL modules with a name starting with a, b, or c.

```
delete_icl_modules {mychip tap tdr}
delete_icl_modules -all
delete_icl_modules {[get_icl_modules {[abc].*}} -regexp]
```

Related Topics

[get_icl_modules](#)

[read_icl](#)

delete_icl_ports

Context: patterns -ijtag, dft

Mode: setup, insertion

Undoes the effect of the add_icl_ports command on a specified list of top level ports.

Usage

`delete_icl_ports port_objects [-silent]`

Description

Undoes the effect of the [add_icl_ports](#) command on a specified list of top level ports.

Arguments

- *port_objects*

A required string that specifies a Tcl list of one or more design port names or a collection of one or more design or icl ports.

The ports in the collection must not have been used as arguments of the -source switch of an add_icl_port command before. (A port which is already in use as a “source port”, cannot be deleted by means of delete_icl_ports.)

- -silent

An optional switch that specifies to suppress the warning messages that are normally generated when the specified port names do not exist or were never the target of previous [add_icl_ports](#) commands.

Examples

The following example declares then deletes ICL DataInPort D1, and ICL DataOutPorts O1 O2 and O3.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
delete_icl_ports {O1 O2 O3}
```

Related Topics

[add_icl_ports](#)

delete_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that will be created during ICL extraction.

Usage

```
delete_icl_scan_interfaces name_list | -All
```

Description

Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that will be created during ICL extraction.

You must set the current design with set_current_design before using the delete_icl_scan_interfaces command.

Arguments

- ***name_list***

A list that specifies the names of scan interfaces that ICL extraction will delete from the new top ICL module.

- **-All**

A switch that specifies that all scan interfaces will be deleted from the new ICL top module.

Examples

The following example will delete the ScanInterfaces named TAP and Internal from the new ICL top module during ICL extraction:

```
delete_icl_scan_interfaces {TAP Internal}
```

Related Topics

[add_icl_scan_interfaces](#)

[get_icl_scan_interface_list](#)

[get_icl_scan_interface_port_list](#)

[set_icl_scan_interface_ports](#)

delete_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Deletes exceptions to Iddq restrictions added with the [add_iddq_exceptions](#) command.

Usage

`delete_iddq_exceptions {original_string... | -All}`

Description

Deletes exceptions to Iddq restrictions added with the [add_iddq_exceptions](#) command.

For more information about IDDQ, see “[IDDQ Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- ***original_string***

A required repeatable string that identifies a site to which you previously applied Iddq exception. The original string is reported by the [report_iddq_exceptions](#) command and must be one of the following:

- A string originally input to an [add_iddq_exceptions](#) command
- A string generated automatically by the “[analyze_restrictions -Int_float_auto_except](#)” command.

- **-All**

A required switch that specifies the deletion of all Iddq restrictions.

Examples

The following example removes an Iddq exception:

`delete_iddq_exceptions /cpu/alu/U2392/`

Related Topics

[analyze_restrictions](#)

[set_iddq_checks](#)

delete_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.

Usage

```
delete_ijtag_logical_connection  
  -from source_pin_or_port_name | -to destination_pin_or_port_name
```

Description

Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.

You must specify the source pin/port or the destination pin/port or both with this command. If both the source and destination pins/ports are specified, the command removes the logical connection between them. If only the source pin/port is specified, then all logical connections that have that pin/port as the source will be removed. If only the destination pin/port is specified, then all logical connections that have that pin/port as the destination will be removed.

Arguments

- **-from *source_pin_or_port_name***

A switch and value pair that specifies the source pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to *destination_pin_or_port_name***

A switch and value pair that specifies the destination pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

Examples

The following example will add three logical connections in the current design, then delete one logical connection.

```
add_ijtag_logical_connection -from /i2/in -to /i2/out  
add_ijtag_logical_connection -from /i3/in -to /i3/out  
add_ijtag_logical_connection -from /i4/in -to /i4/out  
report_ijtag_logical_connections
```

```
// IJTAG Logical Connections:  
// From Source To Destination Inversion  
// ====== ====== ======  
// /i2/in     /i2/out      no  
// /i3/in     /i3/out      no  
// /i4/in     /i4/out      no  
  
delete_ijtag_logical_connection -from /i4/in -to /i4/out  
  
report_ijtag_logical_connections  
  
// IJTAG Logical Connections:  
// From Source To Destination Inversion  
// ====== ====== ======  
// /i2/in     /i2/out      no  
// /i3/in     /i3/out      no
```

Related Topics

[add_ijtag_logical_connection](#)

[report_ijtag_logical_connections](#)

delete_input_constraints

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Removes previously applied constraints from primary input pins.

Usage

Context: dft-scan and dft -test_points

```
delete_input_constraints
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs |
-EQuivalent equivalent_pin...] [constraint_switch]
```

Context: patterns -scan

```
delete_input_constraints
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs |
-EQuivalent equivalent_pin...] [constraint_switch] [-HOLD] [-NO_Z] [-SLOW_pad] [-CELL
model_name]]
```

Context: patterns -ijtag

```
delete_input_constraints
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude| -EQuivalent
equivalent_pin...] [constraint_switch]
```

Description

Removes previously applied constraints from primary input pins.

The delete_input_constraints command removes constraints you previously applied to primary input pins using the [add_input_constraints](#) command.

Arguments

- **-ALl**
An optional switch that specifies to remove all previously applied input constraints, including pin equivalents. This is the default. Constraints for power and ground pins that are added via attributes are not deleted. To remove only pin equivalents, use the -Equivalent switch.
- ***primary_input_pin_name***
An optional, repeatable string that specifies the name of a primary input pin from which to remove the constraint.
- **-BIDI_Only**
An optional switch that specifies to remove previously applied constraints only from bidirectional pins.

- **-BIDI_Exclude**
An optional switch that specifies to remove previously applied constraints only from primary input pins that are *not* bidirectional.
- **-SCAN_INputs**
An optional switch that specifies to remove previously applied constraints only from scan input pins.
- **-EQuivalent *equivalent_pin***
An optional switch and repeatable string that specify the name of a primary input pin from which to remove a previously defined equivalence specification.
- ***constraint_switch***
An optional switch that specifies a particular constraint to remove. The choices, from which you can specify only one, are as follows:

-C0 — Constant 0
-C1 — Constant 1
-CX — Constant X (unknown)
-CZ — Constant Z (high-impedance)
-CT0 — Constant TIE0
-CT1 — Constant TIE1
-CTZ — Constant TIEZ

Tip

 For additional information about the constraints associated with the preceding switches, refer to the same switches under the [add_input_constraints](#) command.

- C_All — All C-type constraints
- C_Default — All C-type constraints added using any of the following switches with the [add_input_constraints](#) command: -All, -Bidi_only, -Bidi_exclude, -Scan_inputs. This does not include C-type constraints added by specifying the primary input pin name explicitly.
- C_Specified — C-type constraints added by specifying the primary input pin name explicitly.
- **-HOld**
An optional switch that specifies to remove the “hold” restriction, if applicable, from the specified pins. See the description of the -Hold argument to the [add_input_constraints](#) command for more information.

- **-NO_Z**

An optional switch that specifies to remove the “no-Z” restriction, if applicable, from the specified pins. See the description of the **-No_z** argument to the **add_input_constraints** command for more information.

- **-Slow_pad**

An optional switch that specifies to remove the “slow-pad” restriction, if applicable, from the specified pins. See the description of the **-Slow_pad** argument to the **add_input_constraints** command for more information.

- **-CELL *model_name***

An optional switch and string pair that, together with the **-Slow_pad** switch, specifies to prefix the instance name of each instance of the Tessent Cell library model of type *model_name* to the *primary_input_pin_name*, and look up each resulting name as the pin directly driven by the primary input.

Examples

The following example deletes constraints from two primary inputs.

```
delete_input_constraints indata2 -c1  
delete_input_constraints indata4 -c0
```

Related Topics

[add_input_constraints](#)

[report_input_constraints](#)

delete_instances

Context: all contexts

Mode: insertion

Deletes the specified instance objects from the current design.

Usage

`delete_instances obj_spec [-silent]`

Description

Deletes the specified instance objects from the current design.

The *obj_spec* can be a Tcl list of one or more hierarchical instance names, or a collection of one or more instances.

Arguments

- *obj_spec*
A required value that specifies one or more instances to delete. The value is a Tcl list of one or more hierarchical instance names, or a collection of one or more instances.
- -silent
An optional argument that specifies to not generate an error messages when an instance in the *obj_spec* list does not exist.

Examples

Example 1

The following example deletes the two specified instances in the Tcl list:

`delete_instances {u1/u2 u1/u4}`

Example 2

The following example deletes the collection of instances returned by the `get_instances` command:

`delete_instances [get_instances ABC* -hierarchical -of_module XYZ]`

Related Topics

[create_instance](#)

[get_common_parent_instance](#)

[rename_instance](#)

[replace_instances](#)

[uniquify_instances](#)

delete_iprocs

Context: all contexts

Mode: all modes

Deletes the specified list of iProcs attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by *module_name*.

Usage

```
delete_iprocs iproc_list [-of_module module_name] [-silent]
```

Description

Deletes the specified list of iProcs attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by *module_name*.

All iProcs attached to the ICL module are deleted when *iproc_list* is not specified.

Arguments

- *iproc_list*
A Tcl list of iProc names.
- *-of_module module_name*
A required switch and value pair that constrains the command to delete only instances of iProcs attached to the module specified by *module_name*.
- *-silent*
An optional switch that specifies to suppress error messages if the specified iProcs or module do not exist.

Return Values

None

Examples

The following example deletes the test1 iProc from module mychip and then searches and deletes any remaining iProcs on mychip that begin with “p”.

```
delete_iprocs test1 -of_module mychip  
delete_iprocs [lsearch -inline -all [get_iproc_list p*] -of_module mychip]
```

Related Topics

[get_iproc_argument_default](#)
[get_iproc_argument_list](#)
[get_iproc_body](#)
[get_iproc_list](#)

[iProc](#)

[iProcsForModule](#)

delete_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, deletes all core instance data or the core instance data from a specified design.

Usage

```
delete_layout_core_information {-chip_design_name name | -all}
```

Description

In hierarchical layout-aware diagnosis, deletes all core instance data or the core instance data from a specified design.

To use this command you must have a valid LDB that you previously opened using the open_layout command.

See “[Diagnosis for Hierarchical Design](#)” in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- **-chip_design_name *name***

A switch and string pair that specifies to delete the core instance data from the specified design.

- **-all**

A switch that specifies to delete all the core instance data.

Related Topics

[add_layout_core_information](#)

[report_layout_core_information](#)

delete_layout_verification

Context: patterns -scan_diagnosis

Mode: analysis

Reports or deletes stored layout verification information from the current layout database.

Usage

```
delete_layout_verification {-List | -Design design_index_in_list}
```

Description

Reports or deletes stored layout verification information from the current layout database.

Arguments

- **-Design *design_index_in_list***

A required switch and integer pair that specifies the layout verification data the tool deletes.
Use the -list switch to print index numbers for all stored design layout verification information.

- **-List**

A required switch that directs the tool to list all the designs and corresponding layout rule violation information stored in the layout database. Each design is listed with its assigned Index number, which is used as an argument to the -design switch. The -list switch does not delete any layout verification information, use the -design switch to delete layout verification data from the database.

Examples

In the following example, the diagnosis lists the layout verification and design information stored in the open_layout database:

```
delete_layout_verification -list

// Note: Layout verification information corresponding to the following
//        designs can be deleted from the layout database
//
//        Index      Stored Violation Count  Flat model file name
//        1.          611                  my_design.flat
//        2.          643                  other_design_version.flat
//
//        Note: The current design is my_design.flat (Index 1.)
//        Note: To delete any of the above, please use
//              delete_layout_verification -design <design index in the list>
//
```

In the following example, the diagnosis deletes the layout verification information for design index 1 (my_design.flat):

```
delete_layout_verification -design 1
```

```
// Note: Deleting layout verification information corresponding
//        to my_design.flat.
// Warning: Since the layout verification information for the
//           current design has been deleted, the layout will be closed.
```

Related Topics

[create_layout](#)

[open_layout](#)

delete_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).

Usage

`delete_lfsr_connections primary_pin... | -All`

Description

Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).

The `delete_lfsr_connections` command deletes the connections between the LFSRs and the primary pins specified with the `add_lfsr_connections` command. You can use the `report_lfsr_connections` command to display all the current connections between LFSRs and primary pins.

Arguments

- ***primary_pin***

A repeatable string that lists the primary pins whose connections to LFSRs you want to delete.

- **-All**

A switch that deletes all of the connections between LFSRs and primary pins.

Examples

The following example changes the definition of an LFSR connection by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_connections scan_in.1 lfsr1 2
delete_lfsr_connections scan_in.1
add_lfsr_connections scan_in.2 lfsr1 2
```

Related Topics

[add_lfsr_connections](#)

[report_lfsr_connections](#)

[delete_lfsr_taps](#)

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the tap positions from a Linear Feedback Shift Register (LFSR).

Usage

`delete_lfsr_taps lfsr_name {tap_position... | -All}`

Description

Removes the tap positions from a Linear Feedback Shift Register (LFSR).

The `delete_lfsr_taps` command deletes the specified LFSR tap positions added with the `add_lfsr_taps` command. To display the current tap positions of all defined LFSRs, use the `report_lfsrs` command.

Arguments

- *lfsr_name*

A string that specifies the reference name of the LFSR whose tap positions you want to delete.

- *tap_position*

A repeatable string that specifies the list of tap positions that you want to delete from the *lfsr_name*.

- **-All**

A switch that deletes all of the tap positions from the *lfsr_name*.

Examples

The following example changes an LFSR tap position by deleting it and then adding a new tap position:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 Prpg 5 13 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_taps lfsr2 1 3
delete_lfsr_taps lfsr1 3
add_lfsr_taps lfsr1 1
```

Related Topics

[add_lfsr_taps](#)

[report_lfsrs](#)

[set_lfsrs](#)

delete_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the specified Linear Feedback Shift Registers (LFSRs).

Usage

`delete_lfsrs lfsr_name... | -All`

Description

Removes the specified Linear Feedback Shift Registers (LFSRs).

The `delete_lfsrs` command deletes LFSRs defined with the `add_lfsrs` command. You can use the `report_lfsrs` command to display a list of the current LFSRs with their current values and tap positions. When you delete an LFSR, the tool also deletes all its taps and pin connections.

Arguments

- *lfsr_name*

A repeatable string that specifies the reference names of the LFSRs that you want to remove.

- **-All**

A switch that deletes all defined LFSRs.

Examples

The following example changes the definition of an LFSR by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs lfsr3 prpg 5 11 -parallel -out
delete_lfsrs lfsr3
add_lfsrs lfsr3 prpg 5 11 -parallel -in
```

Related Topics

[add_lfsrs](#)

[report_lfsrs](#)

[set_lfsrs](#)

delete_lists

Context: dft -edt, patterns -scan

Mode: analysis

Removes pins from the pin list the tool monitors and reports on during simulation.

Usage

delete_lists *pin_pathname*... | -All

Description

Removes pins from the pin list the tool monitors and reports on during simulation.

The **delete_lists** command removes from the monitored pin list, pins that you previously added to the list using the [add_lists](#) command. To review the current list of pins, use the [report_lists](#) command. To put additional pins on the list, use the [add_lists](#) command.

Arguments

- ***pin_pathname***
A repeatable string that specifies the pins you want to remove from the monitored pin list.
- **-All**
A switch that removes all currently listed pins from the monitored pin list.

Examples

The following example removes an extra added output pin from the monitored pin list:

```
add_lists /i_1006/o /i_1007/o /i_1008/o  
report_lists
```

```
3 pins are currently monitored.  
/i_1006/o  
/i_1007/o  
/i_1008/o
```

```
delete_lists /i_1007/o  
report_lists
```



```
2 pins are currently monitored.  
/i_1006/o  
/i_1008/o
```

Related Topics

[add_lists](#)

[report_lists](#)

[set_list_file](#)

delete_loadboard_loopback_pairs

Context: patterns -ijtag, -scan, -scan_diagnosis, -scan_retargeting

Modes: Setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Deletes any loopback connection associated with the specified ports.

Usage

`delete_loadboard_loopback_pairs -all | port_object_spec`

Description

You use this command to delete the loopbacks associated with specified ports or all of the loopbacks that were created with the [add_loadboard_loopback_pairs](#) command.

Arguments

- **-all**
A switch that specifies to delete all loopbacks.
- ***port_object_spec***
A required string that lists the ports from which to delete associated loopbacks. You may specify one or both ports of the associated loopback pairs.

Examples

Example 1

This example shows one loopback pair being deleted using the *port_object_spec*:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5]
dout[0]}report_loadboard_loopback_pairs

-----
Inputs   Outputs   AC Time to Z
-----  -----
din[4]   dout[5]  N/A
ue       dout[0]  N/A
-----

delete_loadboard_loopback_pairs {dout[5] din[4]}report_loadboard_loopback_pairs

-----
Inputs   Outputs   AC Time to Z
-----  -----
ue       dout[0]  N/A
-----
```

Example 2

This example shows how you delete all loopback pairs using the **-all** switch:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5]  
dout[0]}report_loadboard_loopback_pairs
```

```
-----  
Inputs   Outputs   AC Time to Z  
-----  
din[4]   dout[5]  N/A  
ue       dout[0]  N/A  
-----
```

```
delete_loadboard_loopback_pairs -allreport_loadboard_loopback_pairs
```

```
delete_loadboard_loopback_pairs -all  
report_loadboard_loopback_pairs
```

```
-----  
Inputs   Outputs   AC Time to Z  
-----  
-----
```

delete_misrs

Context: pattern -scan

Mode: setup, analysis

Deletes MISRs you specify with the add_misrs command.

Usage

```
delete_misrs {-all | misr_name}
```

Description

Deletes MISRs you specify with the [add_misrs](#) command.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- **-all**

A required switch that specifies removing all MISRs.

- **misr_name**

A required repeatable string that specifies the name of the MISR.

Examples

This example deletes a previously added MISR called misr_0, and then re-adds the MISR as blk1_misr.

```
add_misrs misr_0 -size 24 -seed 000000 -taps 21 22 24
delete_misrs misr_0
add_misrs blk1_misr -size 24 -seed 0000 -taps 21 22 24
report_misrs -all
list of MISRs
blk1_misr  MISR  length=24  in_stream_taps=20,21,23
blk1_misr_pre_unload_value = 00000000000000000000000000000000
```

Related Topics

[add_misrs](#)

[report_misrs](#)

delete_multicycle_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes definitions of multicycle paths previously read in using the `read_sdc` command.

Usage

```
delete_multicycle_paths [-All] [-Verbose]
[-Cycles integer [-Start | -End ]]
[-From source_node... | -FROM_Clock clock_signal_name... ]
[-TO sink_node... | -TO_Clock clock_signal_name... ]
[{-THrough through_node...}...]
```

Description

Removes definitions of multicycle paths previously read in using the `read_sdc` command.

The `delete_multicycle_paths` command deletes from the tool's internal database, multicycle path definitions previously read in by the `read_sdc` command.

Arguments

Tip

 When using the following arguments, you can include asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- **-All**
An optional switch that specifies to delete all multicycle path definitions.
- **-Verbose**
An optional switch that directs the tool to display the definition of each deleted multicycle path as it is deleted.
- **-Cycles *integer* [-Start | -End]**
An optional switch and integer value that define the multicycle paths to delete when the path's cycle count is equal to the value specified by *integer*.
[-Start | -End]
These argument options for the -Cycles switch, specify the cycle count is associated with the clock of the start point (-from) or the end point (-to). The default, -End, is uses the end point clock (-to).
- **-From *source_node...***
An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input

(PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

If a specified source node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- **-FROM_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- **-TO *sink_node*...**

An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

If a specified sink node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

Note

 For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “`delete_multicycle_paths... -to CLK2`” would delete_multicycle_paths that have an end point at D2.

- **-TO_Clock *clock_signal_name*...**

An optional switch and repeatable string pair that specifies the end point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- **-T**hrough *through_node...*

An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified multicycle path. For example, the following multicycle path specification:

```
delete multicycle paths -from G -through H I -through J
K -to L
```

specifies all paths that start from G, pass through either H or I, then pass through J or K, and end at L. If you do not use the -Through argument when specifying a path, the tool will consider all paths from the specified starting points to the specified end points to be false paths.

If a specified through node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

Examples

The following example displays current multicycle path definitions, then deletes one of them, echoing its definition as it is deleted:

```
report multicycle paths -all
// Multicycle Path -from /my_design/a -through /my_design/u1 -cycles 2
// Multicycle Path -to /my_design/c -cycles 3
// Multicycle Path -through /my_design/u24 /my_design/u25 -cycles 2
// Total reported multicycle paths = 3

delete multicycle paths -verbose -to /my_design/c
// Multicycle Path -to /my_design/c (/my_design/c/Y (130)) -cycles 2
// Deleted 1 multicycle path.
```

Refer to the [add_false_paths](#) command for examples of signal matches for pathnames specified using the asterisk (*) wildcard character.

Related Topics

[add_false_paths](#)
[delete_false_paths](#)
[read_sdc](#)
[report_false_paths](#)
[report_multicycle_paths](#)

delete_nets

Context: all contexts

Mode: insertion

Removes net objects inside an instance of a design module.

Usage

`delete_nets obj_spec [-silent]`

Description

Removes net objects inside an instance of a design module.

You cannot delete a net that has a fanin or fanout. You can delete only dangling nets.

You cannot delete a net that is also a pin of an instance. You can delete the corresponding input pin using the `delete_pins` command, but the net will remain if it has objects in its fanout. You can delete the corresponding output pin using the `delete_pins` command, but the net will remain if it has objects in its fanin.

In general, you cannot delete a net that is driven by any RTL construct; however, an exception is that you can delete a dangling net driven by a Verilog assign statement. When you delete such a net, you also delete the assign statement.

Arguments

- ***obj_spec***

A required value that specifies a Tcl list of one or more net names or a collection of one or more net objects as returned by the `get_nets` command. If the specified net object is part of a bus net, the entire bus is deleted.

- **-silent**

An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Examples

Example 1

The following example creates a connection between two pins using a specific net name. Once created, the net cannot be deleted until it is no longer used in any connection.

```
create_connections INSTA/I1 INSTB/O1 -net_name mynet  
get_nets mynet  
{mynet}  
delete_net mynet
```

```
// Error: Cannot remove net 'mynet' in scope 'mychip' as it is being used
// in expression(s).
// Error: Error on 'delete_nets' at 'mynet'.
```

```
delete_connections { INSTA/I1 INSTB/O1}
```

```
delete_net mynet
```

```
get_nets mynet -silent
```

```
{ }
```

Example 2

The following example checks for nets inside instance u1 with no fanin, disconnects their fanout and deletes them.

```
foreach_in_collection net [get_nets u1/*] {
    if {[sizeof_collection [get_fanins $net]] == 0} {
            delete_connections [get_fanouts $net -stop_on pin]
            delete_nets $net
    }
}
```

Related Topics

[create_net](#)

[delete_pins](#)

[get_nets](#)

[get_icl_pins](#)

[get_icl_ports](#)

delete_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the nofault settings from either the specified pin or instance/module pathnames.

Usage

```
delete_nofaults {-All | modulename ... -Module [-PIN module_pin.pathname ...] |
    object_expression ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}]
    [-Class {System | User | Full}] [-VERBose] [(> | >>) file.pathname]
```

Description

Removes the nofault settings from either the specified pin or instance/module pathnames.

The delete_nofaults command deletes nofault settings you added using the add_nofaults command.

You can optionally specify nofault settings that have a specific stuck-at or transition value. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a stuck-at (transition) value when deleting a nofault setting, the command deletes both stuck-at (transition) nofault settings.

You can use the report_nofaults command to display all the current nofault settings.

You can also optionally specify nofault settings that have a specific class code: user-defined, system netlist, or both. If you do not specify a class code, then the command deletes the nofault setting from the user class.

Arguments

- **-All**
A switch that deletes all nofault settings.
- **modulename**
A repeatable string that specifies the name(s) of the module(s) from which you want to delete nofault settings. You must include the -Module switch when you specify a module name.
- **-Module**
A switch that specifies interpretation of the *modulename* argument as a module pathname. All instances of these modules are affected. You can use the asterisk (*) and question mark (?) wildcards for the **modulename** argument, and the tool deletes the nofault for all matching modules or library models.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the

instances with the given module name and deletes nofaults for every pin specified in module_pin.pathname list.

- ***object_expression***

A string representing a list of pathnames of instances or pins from which you want to delete nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- **-Pin**

An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then delete nofault settings from all the pins matched.

- **-Instance**

An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then delete nofault settings from all boundary and internal pins of the instances matched.

- **-Stuck_at 01 | 0 | 1**

An optional switch and literal pair that specifies the stuck-at or transition values from which you want to remove a nofault setting. The choices are as follows:

01 — A literal specifying that for stuck-at faults the tool delete both “stuck-at-0” and “stuck-at-1” nofault settings; or for transition faults the tool delete both “slow-to-rise” and “slow-to-fall” nofault settings. This is the default.

0 — A literal that deletes only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

1 — A literal that deletes only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- **-Class User | System | Full**

An optional switch and literal pair that specifies the source (or class) of the nofault settings which you want to delete. The valid literals are as follows:

User — A literal that deletes the user-entered nofault settings. This is the default.

System — A literal that deletes netlist-based nofault settings.

Full — A literal that deletes all the nofault settings in the user and system classes.

- **-VERbose**

By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

```
// Note: Adding faults for 330 fault sites.
```

When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- **>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example deletesnofault settings from the pins of instance i_1007 and then adds all faults to the circuit, thereby allowing the tool to add_faults to the pins of the i_1007 instance:

```
add_nofaults i_1006 i_1007 i_1008 -instance
delete_nofaults i_1007 -instance
set_system_mode analysis
add_faults -all
```

Related Topics

[add_faults](#)
[add_synchronous_clock_group](#)
[delete_faults](#)
[report_faults](#)
[report_nofaults](#)

delete_nonscan_instances

Context: dft, dft -scan, dft -test_points

Mode: setup

Resets any user-specified `is_non_scannable` attribute on the specified objects.

Usage

```
delete_nonscan_instances -INStances instance_spec | -Modules module_spec | -scan_elements  
    scan_element_spec | -All | -library_models
```

Description

Resets any user-specified `is_non_scannable` attribute on the specified objects, thus stopping to exclude them from the scan insertion process.

The `delete_nonscan_instances` command resets the user-specified `is_non_scannable` attribute (see [add_nonscan_instances](#)) on the specified instances, all instances within the specified modules, the specified scan elements, or all instances. Please note that if the `is_non_scannable` attribute was set for any other reason(s), it will remain set after this operation.

Arguments

- **-INStances *instance_spec***
An optional switch and value pair that causes the tool to delete the specified instances from the non-scan instance list. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also deleted from the non-scan instance list.
- **-Modules *module_spec***
An optional switch and value pair that causes the tool to delete all instances within the specified modules from the non-scan instance list. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tesson library model.
- **-scan_elements *scan_element_spec***
An optional switch and value pair that causes the tool to ignore all instances of the specified scan elements during scan insertion. *scan_element_spec* can be a Tcl list of one or more scan elements or a collection of one or more scan elements.
- **-All**
An optional switch that specifies to delete all instances from the non-scan instance list.
- **-library_models**
An optional switch that causes the tool to delete all nonscan_instances that were marked as such by using the -module switch followed by a Tesson library model name. This switch serves as a replacement for "delete_(non)scan_models -all" since this command is obsolete and should no longer be used.

Examples

The following example deletes an extra sequential non-scan instance called i_1007.

```
add_nonscan_instances -instances i_1006 i_1007 i_1008  
delete_nonscan_instances -instances i_1007  
insert_test_logic
```

Related Topics

[add_nonscan_instances](#)

[report_seq_transparent_procedures](#)

delete_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Allows test logic insertion in the specified regions.

Usage

```
delete_notest_points {{pin.pathname... | instance.pathname... | instance.expression}} |  
-ALL | {-Path critical.pathname} | -ALL_Paths}
```

Description

Allows test logic insertion in the specified regions.

The delete_notest_points command is used to allow test logic insertion in regions of the design that were previously excluded from use because the add_notest_points command was used, or the underlying no_control_point/no_observe_point attributes were set directly. This command works by setting the no_control_point and no_observe_point attributes to false (which allows test logic insertion). This means a simple scenario such as preventing test points everywhere in the design except for one particular block can be done as follows:

```
add_notest_points /  
delete_notest_points /block1
```

One obvious problem is that the no_control_point and no_observe_point attributes actually have three values (true, false, and unspecified). Simply setting the attribute for the top level instance to false when “delete_notest_points -all” is issued will not produce the desired result. Instead, the -all switch will actually set the no_control_points and no_observe_points attributes to unspecified for all instances and pin_pathnames within the design. This also occurs when you use “delete_notest_points /”.

For more information, refer to [Attribute Inheritance Behavior](#) in Chapter 6.

Arguments

- ***pin.pathname***

A repeatable string that specifies a list of pins on which you want to allow test logic insertion.

- **-All**

A switch that allows test logic insertion in the entire netlist by setting the no_control_point and no_observe_point attributes to unspecified for all pin_pathnames, instance_pathnames, and instance_expressions.

- ***instance.pathname***

A required repeatable string that allows test logic insertion on or inside the specified instance by setting the no_control_point and no_observe_point attributes to false for this instance.

Note

 The no_control_point and no_observe_point attribute for the gate-pins inside this instance are NOT affected in any way.

- ***instance.expression***

A string representing a list of instances within the design. The string *instance.expression* is defined as:

```
{ string | string * } ...
```

The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

This argument does not support pathnames to objects below the instance level of a Tessent Cell library model. You can use a pathname expression to select several instances and the tool will then allow test logic insertion for all the pins on those instances; but if the expression specifies a location below the instance level of a Tessent Cell library model, the tool will issue an error message.

- **-Path *critical.pathname* (dft -test_points context only)**

A required switch and name pair that specifies to delete the named critical path. You can list the names of the critical paths using the [report_notest_points](#) command with the -Paths switch. For more information on the format of the file, refer to “[The Path Definition File](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-ALL_Paths**

A required switch that specifies to delete all critical paths.

Examples

Example 1

The following example deletes an incorrect notest circuit point and corrects it with a new circuit point before performing testability analysis:

```
set_system_mode analysis
add_notest_points tr_i ts_i
delete_notest_points tr_i
add_notest_points tr_io
```

Example 2

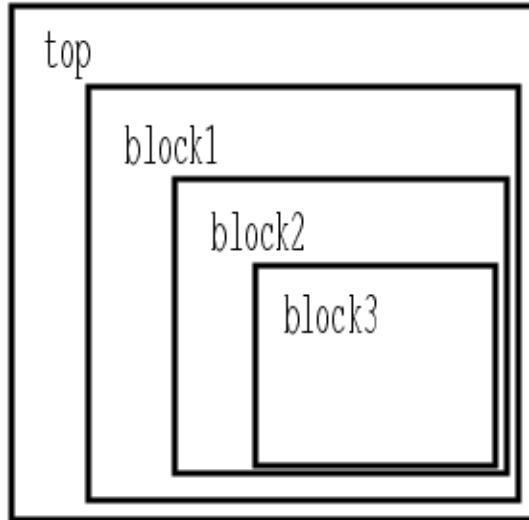
You must be careful when using the add and delete_notest_point commands in conjunction with introspection, as shown in the following example:

```
add_notest_points [get_instances -below_instance /block1]
```

delete_notest_points /block1/block2]

In this example, the `get_instances` command will return `block3`, which is the only instance inside `block1` and `block2`.

Figure 3-26. Hierarchical Test Logic Insertion Example



The attributes will be explicitly set for each of these low-level objects. However, the `delete_notest_points` command will simply set the attribute for the instance `block2` to false. Since the attributes for `block1/block2` are deleted, the tool will insert test logic in the gates/pins/nets inside `block2`. However, any other module that is instantiated below `block2` (`block3` in this example) will remain out of bounds when inserting test logic. This is because the attribute was set specifically for all the instances below `/block1`. If you want to insert test logic for `block3`, then you should instead use the following command:

delete_notest_points [get_instances -below /block1/block2]

Related Topics

[add_output_masks](#)

[report_notest_points](#)

delete_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Removes the masking of the specified primary output pins.

Usage

`delete_output_masks primary_output... | -All`

Description

Removes the masking of the specified primary output pins.

The tools use primary output pins as the observe points during the fault detection process. When you mask a primary output pin with the `add_output_masks` command, the tools mark that pin as an invalid primary output during the fault detection process.

You can set a default mask for all output and bidirectional pins using the `set_output_masks` command. You can add a hold value to a default mask with the `add_output_masks` command, or remove a hold value using the `delete_output_masks` command. To turn off the default masks for all output pins, you must use the “`set_output_masks Off`” command.

Note

 The `delete_output_masks` command can only be used to delete pin constraints added in the same mode. Pin constraints added in setup mode cannot be deleted in analysis mode.

Arguments

- ***primary_output***

A repeatable string that specifies the names of the primary output pins that you want to unmask.

- **-All**

A switch that unmasks all primary outputs masked using the `add_output_masks` command.

Examples

The following example first incorrectly chooses two of the design’s primary output pins to mask. The example then unmasks the one primary output that was inappropriate, masks the correct primary output, and then displays the complete list of currently-masked primary output pins no longer available as observation points:

```
add_output_masks q1 qb3
delete_output_masks q1
add_output_masks qb1
report_output_masks
```

TIEX qb1
TIEX qb3

Related Topics

[add_output_masks](#)
[report_output_masks](#)
[set_output_masks](#)

delete_patterns

Context: dft -edt, patterns -scan, patterns -scan_retargeting,
patterns -scan_diagnosis, patterns -ijtag

Mode: analysis

Deletes the current pattern set.

Usage

Context: patterns -scan

delete_patterns [-external | -internal] [-pattern_sets [chain] [scan]] [-keep_fault_status] [-force]

Context: patterns -ijtag

delete_patterns [-pattern_sets *pattern_set_list*] [-force]

Context: patterns -scan_retargeting

delete_patterns -core *name* [-mode *name*] [-silent]

Description

Deletes the current pattern set.

This command removes either the internal or external pattern set or both from the current session.

You can only delete generated pattern sets (that is, patterns generated by the tool and not read in) that have been written out using the [write_patterns](#) command, unless you use the -force switch. This restriction prevents you from deleting patterns that you have generated but have not yet written out to disk. In the -ijtag context, you can delete only patterns that have been closed.

Arguments

- -external | -internal (**patterns -scan context only**)

Optional switches that specify which pattern source to remove. By default, the command removes both internal and external pattern sets.

- -pattern_sets [chain] [scan] (**patterns -scan only**)

An optional switch and keyword pair that specifies which pattern type (scan or chain) to remove from the current session. By default, the tool removes both scan and chain patterns.

- -pattern_sets *pattern_set_list* (**-ijtag and patterns -scan_retargeting context only**)

An optional switch and string that specifies a Tcl list of pattern set names that were created using the [open_pattern_set](#) command. When you do not use this switch, the command deletes all pattern sets.

- **-keep_fault_status (-scan context only)**

Optional switch that keeps the current fault status when removing the internal patterns. When deleting the internal patterns, the status of existing faults is reset by default if you do not use this switch.

- **-force**

An optional switch that forces removing the internal patterns even when they are not stored previously. By default, the tool does not allow you to remove the unsaved internal patterns. However, the -force option allows you to delete the unsaved internal patterns without this safeguard. You can always remove the external patterns.

- **-core *name* (patterns -scan_retargeting only)**

A required switch that specifies the cores from which pattern sets are to be deleted. If the core *name* does not exist, the patterns for all cores are deleted in the pattern retargetable phase.

- **-mode *name* (patterns -scan_retargeting only)**

An optional switch that specifies which modes of the core the patterns should be deleted from. If -mode is not specified, all modes of the core are deleted.

- **-silent (patterns -scan_retargeting only)**

An optional switch that specifies not to report patterns as they are deleted. By default, deleted patterns are reported.

Examples

Example 1

The following example removes all pattern sets currently in memory:

```
delete_patterns
```

Example 2 (-scan context)

The following example removes only the chain patterns from the internal pattern set. The -force option suppresses the error that would otherwise occur if you had not already saved the internal patterns using the write_patterns command.

```
delete_patterns -internal -pattern_sets chain -force
```

Example 3 (-ijtag context)

The following example removes only the patterns in the Tcl list named *pattern_list5*:

```
delete_patterns -pattern_sets patterns_list5
```

Related Topics

[read_patterns](#)

[simulate_patterns](#)

[write_patterns](#)

[open_pattern_set](#)

delete_pins

Context: all contexts

Mode: insertion

Removes pin objects from a design module instance.

Usage

```
delete_pins obj_spec [-silent]
```

Description

If the pin to be deleted has an internal connection, the internal nets associated to the pin are kept; otherwise they are removed too.

Note

 You should be aware that deleting one bit of a bus deletes the entire bus.

Arguments

- *obj_spec*

A required value that specifies a Tcl list of one or more pin names, or a collection of one or more pin objects as returned by the `get_pins` command. If the specified pin object is part of a bus net, the entire bus is deleted.

- -silent

An optional argument that specifies to not generate an error message if an object within the specified *obj_spec* does not exist.

Examples

Example 1

The following example deletes the specified pins names. On instance u1, pin A is a scalar port, pin B is a bus pin with indices [3:0], and port C is also a bus port with indices [7:0]. All three pins are completely deleted even though the complete bus range was not specified for pins B and C.

```
get_pins {u1/A u1/B u1/C}
{u1/A u1/B[3] u1/B[2] u1/B[1] u1/B[0] u1/C[7] u1/C[6] u1/C[5] u1/C[4] u1/
C[3] u1/C[2] u1/C[1] u1/C[0]}

delete_pins {u1/A u1/B u1/C[1]}
get_pins {u1/A u1/B u1/C}

// Warning: No result. Please check your filtering options.
{}
```

Example 2

The following example deletes the collection of pin objects returned by the get_pins command.

```
delete_pins [get_pins {u1/A u1/B u1/C[1]} -silent]
```

Related Topics

[create_pin](#)

[get_icl_pins](#)

[get_icl_ports](#)

[get_nets](#)

delete_ports

Context: all contexts

Mode: insertion

Removes port objects from a design module.

Usage

```
delete_ports obj_spec1 [-on_module obj_spec2] [-silent]
```

Description

Removes port objects from a design module.

If the port to be deleted has an internal connection, the internal nets associated to the port are kept; otherwise they are removed too.

If *obj_spec1* is a collection of ports, those names refers to ports on specific modules and the specified *-on_module* option is ignored.

Note

 Deleting one bit of a bus port deletes the entire bus port.

Arguments

- ***obj_spec1***
A required value that specifies a Tcl list of one or more port names or a collection of one or more port objects.
- ***-on_module obj_spec2***
An optional switch and value pair that specifies the module from which the ports are to be deleted. The *obj_spec2* value can be the name of a module or a module object returned by the command. If the *-on_module* switch is not specified, this command deletes the port from the current design which is the module specified by the *set_current_design* command. You cannot delete a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to delete ports on modules created with the *create_module* command.
- ***-silent***
An optional argument that specifies to not generate an error message if an object within the specified *obj_spec1* or *obj_spec2* does not exist.

Examples

Example 1

The following example deletes the specified port names of module ModA. On ModA, port A is a scalar port, port B is a bus port with indices [3:0], and port C is also a bus port with

indices [7:0]. All three ports are completely deleted even if the complete bus range was not specified for port B and C.

delete_ports {A B C[1]} -on_module ModA

Example 2

The following example deletes the collection of port objects returned by the get_ports command. Notice that the -on_module option is not specified for the delete_ports command because the port objects returned by the get_ports command are already associated to a specific module.

delete_ports [get_ports {A B C[1]} -of_module ModA]

Related Topics

[create_port](#)

[get_icl_ports](#)

delete_primary_inputs

Context: all contexts

Mode: setup

Removes the specified primary inputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

Usage

```
delete_primary_inputs {net.pathname... | primary_input_pin... | -All}  
[-Class {User | System | Full}]
```

Description

Removes the specified primary inputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

The delete_primary_inputs command deletes from a circuit the primary inputs that you specify. You can delete either the user class, system class, or full classes of primary inputs. If you do not specify a class, the tool deletes the primary inputs from the user class.

You can display a list of any class of primary inputs by using the report_primary_inputs command.

Note

 Once you delete a primary input pin with this command, adding the pin back again with the add_primary_inputs command will not create the same pin. Bus information is lost and the pin will be marked as user-added. Take care to use delete_primary_inputs only if you wish to truly delete a pin from the design so it is not used.

Arguments

- ***net.pathname***

A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- ***primary_input_pin*...**

A repeatable string that specifies a list of primary input pins that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- **-All**

A switch that deletes all primary inputs. You can specify the class of primary inputs to delete with the -Class switch.

- -Class [User](#) | System | Full

An optional switch and literal pair that specifies the class code of the designated primary input pins. The valid class code literal names are as follows:

[User](#) — A literal specifying that you added the primary inputs using the [add_primary_inputs](#) command. This is the default class.

[System](#) — A literal specifying that the primary inputs derive from the netlist.

[Full](#) — A literal specifying that the primary inputs consist of both user and system classes.

Note

 In dft -scan context and dft -test_points context, the [delete_primary_input](#) -class system or -class full are not allowed.

Examples

The following example deletes an extra added primary input from the user class of primary inputs:

```
add_primary_inputs indata2 indata4 indata6 delete_primary_inputs indata4 -class user
```

Related Topics

[add_primary_inputs](#)

[report_primary_outputs](#)

delete_primary_outputs

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the specified primary outputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

Usage

```
delete_primary_outputs {net.pathname... | primary_output_pin... | -All}
[-Class {User | System | Full}]
```

Description

Removes the specified primary outputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

The delete_primary_outputs command deletes from a circuit the primary outputs that you specify. You can delete either the user class, system class, or full classes of primary outputs. If you do not specify a class, the tool deletes the primary outputs from the user class.

You can display a list of any class of primary outputs by using the report_primary_outputs command.

Note

 Once you delete a primary output pin with this command, adding the pin back again with the add_primary_outputs command will not create the same pin. Bus information is lost and the pin will be marked as user-added. Take care to use delete_primary_outputs only if you wish to truly delete a pin from the design so it is not used.

Arguments

- ***net.pathname***

A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- ***primary_output_pin***

A repeatable string that specifies a list of primary output pins that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- **-All**

A switch that deletes all primary outputs. You can specify the class of primary outputs to delete with the -Class switch.

- -Class User | System | Full

An optional switch and literal pair that specifies the class code of the primary output pins that you specify. The valid literal names are as follows:

User — A literal specifying that the list of primary outputs were added using the add_primary_outputs command. This is the default class.

System — A literal specifying that the list of primary outputs derive from the netlist.

Full — A literal specifying that the list of primary outputs consists of both the user and system class.

Examples

The following example deletes a primary output from the system class of primary outputs:

```
delete_primary_outputs outdata1 -class system
```

Related Topics

[add_primary_outputs](#)

[report_primary_outputs](#)

delete_processors

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion (dft -edt context only)

Removes processor definitions previously specified with the [add_processors](#) command.

Usage

`delete_processors [hostname[:cpus]]...`

Description

Removes processor definitions previously specified with the [add_processors](#) command.

The `delete_processors` command deletes slave processor definitions you previously specified using the [add_processors](#) command. When you issue the command with no arguments, the tool discontinues all current slave processes. If you specify one or more *hostname* arguments, the tool will discontinue slave processes only on the specified host machines. If you include *:cpus* with *hostname*, the tool will discontinue only the specified number of slave processes on that host machine. Without the *:cpus*, all current slave processes on that host machine are discontinued. When specifying *hostname*, you must specify the network name of the machine, even if it was originally selected by an LSF, SGE, or custom job scheduler.

Be aware that the process run by an LSF, SGE, or custom job scheduler is a script rather than the actual slave process. If you discontinue these slave processes with the `delete_processors` command, the corresponding scripts also terminate.

Tip

 Alternatively, you can end execution of the scripts by using the job scheduler commands (SGE's `qdel`, LSF's `bkill`, or whatever is used by the job scheduler invoked by the [add_processors](#) Generic command) or a Linux `kill` command. There will be a small delay before the slave process is actually shut down to allow for the detection of the script death and for the shutdown to be coordinated with the master process. Slave processes shut down in this manner will appear to have died in the master process's transcript. This is normal behavior. Detection of script deaths occurs only during the [add_processors](#), `delete_processors`, and [report_processors](#) commands. You can force detection from the command line without other side effects by issuing the [report_processors](#) command with no arguments.

Arguments

- *hostname*

A repeatable string that specifies a host machine you previously defined with the [add_processors](#) command or received via a job scheduler. The *hostname* choices are as follows:

machine_name — A string that specifies the host machine by its network name.

LOCALHOST — A literal that specifies the host machine is the machine on which you invoked the tool.

IPaddress — A string that specifies the host machine by its IP address.

- **:cpus**

An optional colon (:) and positive integer pair that specifies the number of processes to delete on the particular host machine.

Examples

The following example begins by reporting the distributed ATPG status. It then discontinues all the slave processes currently running on one remote host (nemo) and one of the slave processes running on the other remote host (ahab):

report_processors

hosts/processors	arch	CPU(s)	%idle	free RAM	process size
ace (master)	x86	2 x 2.2 GHz	81%	8393.52 MB	20.07 MB
nemo	1 x86	2 x 2.0 GHz	98%	382.33 MB	18.78 MB
	2				18.78 MB
	3				18.78 MB
	4				18.78 MB
ahab	1 x86	1 x 1.7 GHz	99%	334.98 MB	18.78 MB
	2				18.78 MB

master and 6 processors running.

delete_processors nemo ahab:1

```
// Deleted host nemo:4
// Deleted host ahab:1
// Total processors: 1
// Note: Freeing 1 'Tessent FastScan' license.
```

report_processors

hosts/processors	arch	CPU(s)	%idle	free RAM	process size
ace (master)	x86	2 x 2.2 GHz	81%	8393.52 MB	20.07 MB
ahab	1 x86	1 x 1.7 GHz	97%	229.83 MB	18.78 MB

master and 1 processor running.

Related Topics

- [add_processors](#)
- [compress_patterns](#)
- [create_patterns](#)
- [identify_redundant_faults](#)
- [order_patterns](#)
- [report_processors](#)

[delete_read_controls](#)

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the read control line definitions from the specified primary input pins.

Usage

`delete_read_controls primary_input_pin... | -All`

Description

Removes the read control line definitions from the specified primary input pins.

The `delete_read_controls` command deletes read control lines defined with the `add_read_controls` command. You can delete the read control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***
A repeatable string that specifies a list of primary input pins from which you want to delete any read control line definitions.
- **-All**
A switch that deletes the read control line definitions for all primary input pins.

Examples

The following example deletes an incorrect read control line, then redefines that read control line with the correct off-state:

```
add_read_controls 0 r1 r2
delete_read_controls r1
add_read_controls 1 r1
set_system_mode analysis
```

Related Topics

[add_read_controls](#)
[report_read_controls](#)

delete_register_value

Context: all contexts

Mode: setup

Deletes all or a specified register value variable specified with the [add_register_value](#) command.

Usage

```
delete_register_value {-All | value_name ...}
```

Description

Deletes all or a specified register value variable specified with the [add_register_value](#) command.

See “[Serial Register Load and Unload for LogicBIST and ATPG](#)” in the *Tessent Shell User’s Manual* for complete information.

Arguments

- **-All**
A required switch that deletes all register value variables.
- ***value_name* ...**
A required repeatable string that deletes a named *value_name*.

Examples

The following example deletes all register value variables:

```
delete_register_value -all
```

Related Topics

[add_register_value](#)
[report_register_value](#)
[set_register_value](#)

[delete_rtl_to_gates_mapping](#)

Context: all contexts

Mode: setup

Removes name-mapping rules for RTL to post-synthesis/post-layout name mapping.

Usage

```
delete_rtl_to_gates_mapping -instance_name_map_index instance_name_map_index
| -register_name_map_index register_name_map_index
```

Description

Removes the name-mapping rules for RTL to post-synthesis/post-layout name mapping build in the tool or previously added by the user using the [add_rtl_to_gates_mapping](#) command. Use the [report_rtl_to_gates_mapping](#) command to see the integer index value associated to each mapping.

Arguments

- **-instance_name_map_index *instance_name_map_index***

A switch and string pair that specifies the instance renaming rule index to remove. Use the [report_rtl_to_gates_mapping](#) command to see the integer index value associated to each mapping.

- **-register_name_map_index *register_name_map_index***

A switch and string pair that specifies the register renaming rule index to remove. Use the [report_rtl_to_gates_mapping](#) command to see the integer index value associated to each mapping.

Examples

The following example shows the result of [report_rtl_to_gates_mapping](#) before and after the [delete_rtl_to_gates_mapping](#) command was called. You can see that the built in mapping rule “`%(name)_reg[%(d0)][%(d1)][%(d2)][%(d3)]`” and the user-specified instance mapping “`corea_i1/coreb_i1 -> corea_i1`” rules are deleted by using their respective indexes seen in the report created by the [report_rtl_to_gates_mapping](#) command.

```
report_rtl_to_gates_mapping
```

delete_rtl_to_gates_mapping

```
// Index Register name map
// -----
// 1  '\% (name)_reg '
// 2  '\% (name)_reg_%(d0) '
// 3  '\% (name)_reg[%(d0)] '
// 4  '\% (name)_reg[%(d0)][%(d1)] '
// 5  '\% (name)_reg[%(d0)][%(d1)][%(d2)] '
// 6  '\% (name)_reg[%(d0)][%(d1)][%(d2)][%(d3)] '
// 7  '%(name)_reg_%(d0)'
// 8  '%(name)_reg_%(d0)%'
// 9  '%(name)_reg_%(d0)%%(d1)%'
// 10 '%(name)_reg_%(d0)%%(d1)%%(d2)%'
// 11 '%(name)_reg_%(d0)%%(d1)%%(d2)%%(d3)%'
// 12 'rtlcreg_%(name)_%(d0)'

// Index Original instance name Mapped instance name
// -----
// 1 corea_i1/coreb_i1 corea_i1

delete_rtl_to_gates_mapping -instance_name_map_index 1delete_rtl_to_gates_mapping -register_name_map_index 6 report_rtl_to_gates_mapping

// Index Register name map
// -----
// 1  '\% (name)_reg '
// 2  '\% (name)_reg_%(d0) '
// 3  '\% (name)_reg[%(d0)] '
// 4  '\% (name)_reg[%(d0)][%(d1)] '
// 5  '\% (name)_reg[%(d0)][%(d1)][%(d2)] '
// 6  '%(name)_reg_%(d0)'
// 7  '%(name)_reg_%(d0)%'
// 8  '%(name)_reg_%(d0)%%(d1)%'
// 9  '%(name)_reg_%(d0)%%(d1)%%(d2)%'
// 10 '%(name)_reg_%(d0)%%(d1)%%(d2)%%(d3)%'
// 11 'rtlcreg_%(name)_%(d0)' 
```

delete_scan_chain_families

Context: dft -scan

Mode: analysis

Removes scan chain families.

Usage

```
delete_scan_chain_families {obj_spec | -all} [-force]
```

Description

Use this command to remove scan chain families.

Arguments

- *obj_spec*

An required list that specifies a collection of scan chain families that should be deleted.

- **-all**

An required switch that specifies that all scan chain families that have been added so far should be deleted.

Note

 If included in one or more scan mode, you must first delete these modes using the “delete_scan_modes” command.

- **-force**

An optional switch that deletes associated scan modes along with the specified scan chain families.

delete_scan_chains

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup

Removes the specified scan chain definitions from the scan chain list.

Usage

`delete_scan_chains chain_name... | -All`

Description

Removes the specified scan chain definitions from the scan chain list.

The `delete_scan_chains` command deletes scan chains defined with the `add_scan_chains` command. You can delete the definitions of specific scan chains or of all scan chains.

dft -scan Details

When you remove a scan chain definition, it is only the definition you are removing, not the scan chain itself.

Arguments

- ***chain_name***
A repeatable string that specifies the names of the scan chain definitions that you want to delete.
- **-All**
A switch that deletes all scan chain definitions.

Examples

The following example defines several scan chains, adding them to the scan chain list, then deletes one of the scan chains:

```
add_scan_chains chain1 group1 indata2 outdata4
add_scan_chains chain2 group1 indata3 outdata5
add_scan_chains chain3 group1 indata4 outdata6
delete_scan_chains chain2
```

Related Topics

[add_scan_chains](#)

[report_scan_chains](#)

delete_scan_elements

Context: dft -scan, dft -test_points, dft -no_rtl

Mode: analysis

Removes scan elements (such as chain or segment) of a given name.

Usage

delete_scan_elements {*obj_spec* | -all }

Description

Use this command to remove scan elements (such as chain or segment) of a given name.

Arguments

- ***obj_spec***

An required list that specifies a collection of scan elements that should be deleted.

- **-all**

A required switch that specifies that all scan elements that have been added so far should be deleted.

Note

 Only scan elements created by you in the current system mode can be deleted.

delete_scan_groups

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup

Removes the specified scan chain group definitions from the scan chain group list.

Usage

`delete_scan_groups group_name... | -All`

Description

Removes the specified scan chain group definitions from the scan chain group list.

The `delete_scan_groups` command deletes scan chain groups defined with the `add_scan_groups` command. You can delete the definitions of specific scan chain groups or of all scan chain groups.

When you delete a scan chain group, the tool also deletes all scan chains within the group.

Arguments

- ***group_name***

A repeatable string that specifies the names of the scan chain group definitions that you want to delete.

- **-All**

A switch that deletes all the scan chain group definitions.

Examples

The following example defines two scan chain groups, adding them to the scan chain group list, then deletes one of the scan chain groups:

```
add_scan_groups group1 scanfile1
add_scan_groups group2 scanfile2
delete_scan_groups group1
```

Related Topics

[add_scan_groups](#)

[report_scan_groups](#)

delete_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Removes the specified, sequential instances from the user-identified scan instance list.

Usage

```
delete_scan_instances -INStances instance_spec | -Modules module_spec | -All |  
-library_models
```

Description

Removes the specified, sequential instances from the user-identified scan instance list.

The delete_scan_instances command deletes sequential instances, library models or all instances within the specified module that were previously added to the scan instance list by using the [add_scan_instances](#) command. You can delete either a specific list of instance names or all instances.

You can also use this command to remove an exception status from an instance, that you added using the [add_scan_instances](#) command, which allowed the tool to convert the instance into a scan cell even though it violated the S4 rule. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

User-identified scan instances result from using the add_scan_instances command.

If you issue an insert_test_logic command after removing an instance from the user-identified scan list with the delete_scan_instances command, the tool then has the option of including it in the system-identified scan instance list.

Arguments

- **-INStances *instance_spec***

An optional switch and value pair that causes to tool to delete the specified instances from the user-identified scan instance list. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also deleted from the user-identified scan instance list.

- **-Modules *module_spec***

An optional switch and value pair that causes to tool to delete all instances within the specified modules from the user-identified scan instance list. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tesson library model.

- **-All**
An optional switch that specifies to delete all instances from the user-identified scan instance list. This switch does not affect the instances in the system-identified scan instance list.
- **-library_models**
An optional switch that causes the tool to delete all nonscan_instances that were marked as such by using the -module switch followed by a Tessent library model name. This switch serves as a replacement for “delete_(non)scan_models -all” since this command is obsolete and should no longer be used.

Examples

The following example deletes an extra sequential scan instance that was defined to be treated as a scan cell; thus, the deleted instance is no longer included in the user-identified scan instance list:

```
add_scan_instances -instances i_1006 i_1007 i_1008  
delete_scan_instances i_1007
```

Related Topics

[add_scan_instances](#)
[report_seq_transparent_procedures](#)

delete_scan_modes

Context: dft -scan

Mode: analysis

Removes one or more scan modes.

Usage

```
delete_scan_modes {obj_spec | -all}
```

Description

Use this command to remove one or more chain scan modes.

Arguments

- ***obj_spec***
An required list of scan modes that should be deleted.
- **-all**
An required switch that specifies that all scan modes that have been added so far should be deleted.

delete_simdut_fault

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Removes the injected stuck-at fault at the specified signal or all the injected faults.

Usage

```
delete_simdut_fault {-signal signal_name / -all}
```

Description

Removes the injected stuck-at fault at the specified signal or all the injected faults.

Arguments

- **-signal** *signal_name* / **-all**

A required choice, as follows:

-signal *signal_name*: deletes the specified injected stuck-at fault at the specified signal.

-all: deletes all of the injected faults stored in the simulator.

Examples

The following example removes an injected stuck-at fault from the SIMDUT_TB.DUT_inst.memory00.q_a signal.

```
delete_simdut_fault -signal SIMDUT_TB.DUT_inst.memory00.q_a
```

Related Topics

[add_simdut_fault](#)

[report_simdut_faults](#)

[delete_simulation_contexts](#)

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist

Deletes one or more user-defined simulation contexts.

Usage

```
delete_simulation_contexts {context_name... | -all_user_defined}
```

Description

Deletes one or more user-defined simulation contexts.

The `delete_simulation_contexts` command allows you to delete one or multiple or all user-defined simulation contexts.

Arguments

- ***context_name***

A required string that defines the name of one or more simulation contexts. Note that you cannot use wildcard characters in the `context_name`. You also cannot delete any of the predefined context names listed in [Table 6-7](#) on page 2023.

- **-all_user_defined**

A required switch that specifies all user-defined simulation contexts.

Examples

Example 1

The following example lists all of the user-defined simulation contexts and then deletes them:

```
SETUP> get_simulation_context_list -user_defined {myContext1 myContext2 myContext3}  
SETUP> delete_simulation_contexts -all_user_defined  
SETUP> get_simulation_context_list -user_defined { }
```

Example 2

The following example lists all of the user-defined simulation contexts and then deletes two of them:

```
SETUP> get_simulation_context_list -user_defined {myContext1 myContext2 myContext3}  
SETUP> delete_simulation_contexts { myContext2 myContext3 }  
SETUP> get_simulation_context_list -user_defined { myContext1 }
```

Related Topics

[add_simulation_context](#)

[report_simulation_contexts](#)

delete_simulation_contexts

[copy_simulation_context](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[set_current_simulation_context](#)
[get_current_simulation_context](#)
[simulate_clock_pulses](#)
[get_simulation_value_list](#)
[simulate_forces](#)

delete_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Removes forces from one or more gate_pin objects.

Usage

```
delete_simulation_forces {obj_spec | -all}
```

Description

Removes forces from one or more gate_pin objects.

The delete_simulation_forces command removes simulation forces from the specified gate_pin object(s). When this command removes a simulation force, the tool immediately propagates the values on the driving gate_pin objects to keep the circuit in a consistent state.

This command is allowed for user-defined contexts only.

Arguments

- *obj_spec*
A required value that specifies one or more gate_pin objects.
- **-all**
A required switch that removes all simulation forces on all gate_pin objects within the current simulation context.

Examples

The following example shows the effect of adding a force and then removing all forces:

```
ANALYSIS> set_current_simulation_context myContext
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 1
ANALYSIS> report_simulation_forces
Value      Object
-----  -----
1          I1

ANALYSIS> delete_simulation_forces -all
ANALYSIS> report_simulation_forces
```

Related Topics

[add_simulation_context](#)
[get_simulation_value_list](#)
[add_simulation_forces](#)

[report_simulation_contexts](#)
[copy_simulation_context](#)
[report_simulation_forces](#)
[delete_simulation_contexts](#)
[simulate_clock_pulses](#)
[get_simulation_context_list](#)
[simulate_forces](#)

delete_synchronous_clock_group

Context: dft, patterns

Mode: setup

Removes one or more user-defined synchronous clock groups.

Usage

```
delete_synchronous_clock_group { object_spec | -all }
```

Note

 Synchronous clock groups have no impact on scan chain analysis, test point identification, and EDT IP insertion (in the dft -scan, dft -test_points and dft -edt sub-contexts).

Description

Removes one or more user-defined synchronous clock groups.

The delete_synchronous_clock_group command removes clock groups that you defined with the add_synchronous_clock_group command. For more information about synchronous clock groups, refer to the [add_synchronous_clock_group](#) command.

Arguments

- ***object_spec***

A Tcl list or collection of pin or net objects that specify a current synchronous clock group. You must specify the complete original object_spec for the clock group you want to remove.

- **-all**

A switch that specifies all synchronous clock groups.

Examples

The following example creates two synchronous clock groups and then removes one of them:

```
SETUP> report_clocks
User-defined Clocks (4) :
=====
Sync and Async Source Clocks
=====
-----  -----
Name   Off State Constraints Internal
-----  -----
I1     0          No
I2     0          No
I3     0          No
I4     0          No
```

```
SETUP> add_synchronous_clock_group {I1 I2}
SETUP> report_synchronous_clock_groups
```

```
User-defined Synchronous Clock Group
=====
// No.  Clocks (Labels)
// --- -----
//   1    I1
//     I2
```

```
SETUP> add_synchronous_clock_group {I3 I4}
SETUP> report_synchronous_clock_groups
```

```
User-defined Synchronous Clock Groups
=====
// No.  Clocks (Labels)
// --- -----
//   1    I1
//     I2
//   2    I3
//     I4
```

```
SETUP> delete_synchronous_clock_group {I1}
```

```
// Error: Specified synchronous clock group does not exist.
```

```
SETUP> delete_synchronous_clock_group {I2 I1}
SETUP> report_synchronous_clock_groups
```

```
User-defined Synchronous Clock Group
=====
// No.  Clocks (Labels)
// --- -----
//   1    I3
//     I4
```

Related Topics

[add_clocks](#)
[add_synchronous_clock_group](#)
[get_synchronous_clock_groups](#)
[report_clocks](#)
[report_synchronous_clock_groups](#)

delete_test_points

Context: dft -test_points

Mode: setup

Removes the specified test point definitions.

Usage

```
delete_test_points [-control | -observe] [-location pin/port_spec | -all]
```

Description

Removes the specified test point definitions.

If you do not specify -control or -observe with either the -location *pin/port_spec* or the -all option, the tool removes all the control and observe test points.

This command removes both user-defined and system-defined test points. You can create user-defined test points with the [add_control_points](#) and [add_observe_points](#) commands.

Arguments

- **-control**
An optional switch that removes the control test points from the specified locations.
- **-observe**
An optional switch that removes the observe test points from the specified locations.
- **-location *pin/port_spec***
A switch and string pair that specifies a pathname to the control and/or observe test points to delete. *pin/port spec* can be a Tcl list of one or more pins/ports, or a collection of one or more pin/port objects.
- **-all**
A switch that removes the control and/or observe test points from all locations.

Examples

The following example removes test points:

```
delete_test_points -location {I_6_16/cp I_7_16/q}
```

The `delete_test_points` command only specifies the pathnames of the test points, not the type. This example includes both control and observe test points and deletes them by default.

Related Topics

[add_input_constraints](#)

[add_tied_signals](#)

delete_test_points

[analyze_test_points](#)
[insert_test_logic](#)
[report_test_logic](#)
[report_test_points](#)
[set_test_point_insertion_options](#)
[set_test_point_analysis_options](#)
[set_test_point_types](#)
[write_test_point_dofile](#)

delete_tied_signals

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the assigned (tied) value from the specified floating nets or pins.

Usage

`delete_tied_signals {floating_object_name... | -All} [-Pin]`

Description

Removes the assigned (tied) value from the specified floating nets or pins.

The `delete_tied_signals` command deletes the tied values assigned with the `add_tied_signals` command. You can display a list of tied signals using the `report_tied_signals` command.

Whenever you delete tied values from nets or pins, be sure to re-add any necessary values before performing another simulation. If you do not add required tied values to floating nets or pins, the tool displays a warning. The warning states that the design has floating nets or pins and assumes they are tied to the default value; you must set the default value using the `set_tied_signals` command.

Arguments

- ***floating_object_name***

A repeatable string that specifies the names of the tied floating nets or pins whose tied values you want to delete.

If you do not specify the `-Pin` option, the tool assumes `floating_object_name` is a net name. If you specify the `-Pin` option, it assumes the `floating_object_name` is a pin name.

- **-All**

A switch that deletes the tied values from all tied floating nets or pins.

- **-Pin**

A switch specifying that the `floating_object_name` argument that you provide is a floating pin name.

Examples

The following example deletes the tied value from the user-class tied net “vcc”; thereby leaving “vcc” as a floating net:

```
add_tied_signals 1 vcc vdd
delete_tied_signals vcc
```

Related Topics

[add_tied_signals](#)

[report_tied_signals](#)

[set_tied_signals](#)

[delete_write_controls](#)

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the write control line definitions from the specified primary input pins.

Usage

`delete_write_controls primary_input_pin... | -All`

Description

Removes the write control line definitions from the specified primary input pins.

The `delete_write_controls` command deletes write control lines defined with the `add_write_controls` command. You can delete the write control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***
A repeatable string that specifies a list of primary input pins from which you want to delete any write control line definitions.
- **-All**
A switch that deletes the write control line definitions for all primary input pins.

Examples

The following example deletes an incorrect write control line, then re-adds that write control line with the correct off-state:

```
add_write_controls 0 w1 w2
delete_write_controls w1
add_write_controls 1 w1
set_system_mode analysis
```

Related Topics

[add_write_controls](#)
[report_write_controls](#)

diagnose_failures

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: The [read_patterns](#) command must be set to a pattern source.

Diagnoses the failures in the specified failure file.

Usage

```
diagnose_failures {failure_filename | -Internal} [-Output report_filepath />
    pipe_to_filename] [-Csv csv_filename] [-Replace] [-Faulty_chain {faulty_chain_name
    faulty_chain_type}...]
```

Description

Diagnoses the failures in the specified failure file.

The diagnose_failures command performs a diagnosis of the failing scan and chain test patterns in the specified failure file, or diagnosis of the IDDQ patterns in the specified failure file.

For scan and chain test patterns, you must translate the ATE failure log into a failure file as described in the “[Guidelines for Preparing the ATE Failure File](#)” section in the Tessent Diagnosis User’s Manual. For IDDQ patterns, see “[IDDQ Diagnosis](#)” for the required failure file format.

It is critical that the data in the test patterns, flattened netlist, and failure file be synchronized for diagnosis. For more information, see “[Input File Requirements](#)” in the Tessent Diagnosis User’s Manual.

For complete information on running diagnosis and interpreting the results, see the [Tessent Diagnosis User’s Manual](#).

Arguments

- ***failure_filename***

A required string that specifies the name of the file that contains the failing test pattern information for diagnosis. This file can use either the pattern-based or cycle-based format as described in the “[Guidelines for Preparing the ATE Failure File](#)” section in the Tessent Diagnosis User’s Manual.

For IDDQ diagnosis, specify the IDDQ failure file. See “[IDDQ Diagnosis](#)” for more information.

- **-Internal**

An required switch that directs the tool to use the in-memory failure file. You must issue a [read_failures](#) command before you use the diagnose_failures -internal command. See “[Diagnosis Improvements and Retrieving Internal Scan Cell Information](#)” in the Tessent Diagnosis User’s Manual for complete information.

- -Output *report_filepath / > pipe_to_filename*

An optional choice that directs the tool to write the diagnostic report in ASCII format to the specified file. By default, the diagnostic report displays on screen.

-Output *report_filepath*: An optional switch and string pair that specifies the output file. This option adds the physical layout information to the report. The physical layout information is not included in the transcript.

> *pipe_to_filename*: An optional redirection operator and string pair that specifies the output file. Choose this option when you have specified set_diagnosis_options - internal_failing_cells on. The internal failing cell data is only reported to the transcript unless you redirect it to a file; however, the -output switch does not include the internal failing cell data in its report. See “[Internal Scan Cell Profiling for Compressed Patterns](#)” for more information.

- -Csv *filename*

An optional switch and string pair that directs the tool to write the diagnostic report in Comma Separated Value (CSV) format to the specified file. The CSV format does not include the list of failing patterns for each symptom. By default, the diagnostic report displays on screen.

- -Replace

An optional switch that directs the tool to overwrite existing diagnostic report files in either ASCII or CSV format. By default, the tool will not overwrite existing files.

- -Faulty_chain *faulty_chain_name faulty_chain_type*

An optional switch and repeatable string and literal pair that allows you to run diagnosis on specified chain failures in addition to the scan failures contained in a failure file.

faulty_chain_name — string that specifies the name of the faulty chain.

fault_type — literal that specifies the fault type associated with the faulty chain. Valid fault types include:

STUCK_AT_1

STUCK_AT_0

SLOW_TO_FALL

SLOW_TO_RISE

SLOW

FAST_TO_RISE

FAST_TO_FALL

FAST

INDETERMINATE — specifies the fault type cannot be determined.

Examples

Example 1

The following example: sets *pattern_file1* as the external test pattern source, injects a stuck-at-0 fault into the design, writes out a failure file named *fail_patterns*, and performs a diagnosis of the failing patterns in the *fail_patterns* file.

```
set_system_mode analysis
read_patterns pattern_file1
write_failures fail_patterns i_1006/i1 -stuck_at 0

// failing_patterns=9 simulated_patterns=36 simulation_time=0.00 sec

diagnose_failures fail_patterns

...
```

Still using the external test pattern source file *pattern_file1*, the following example: injects an additional stuck-at-0 fault into the *fail_patterns* failure file, performs a diagnosis of the failing patterns in the *fail_patterns* file, and sends the report to a file named *fail_diags*.

```
write_failures fail_patterns i_1005/i1 -stuck_at 0 -replace

// failing_patterns=4 simulated_patterns=36 simulation_time=0.00 sec

diagnose_failures fail_patterns -output fail_diags
```

Example 2

The following example loads patterns from two files into the tool external pattern database and performs two separate diagnoses using the default settings. The example writes the resulting reports to files instead of displaying them onscreen. The failure file, *scan_failure_file*, contains only a scan test section (“*scan_test*” keyword followed by scan test fails), so the tool performs only a scan diagnosis for that file. The second file, *chain_failure_file*, contains a chain test section (“*chain_test*” keyword followed by chain test fails) as well as a scan test section. The tool performs only a chain diagnosis for that file.

```
read_patterns pat_file1
read_patterns pat_file2 -append
diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv
diagnose_failures chain_failure_file -output chain_diag.ascii -replace
```

Example 3

The following example first performs chain diagnosis on a failure file that contains a *chain_test* section. The example next attempts a diagnosis of a failure file that contains no *chain_test* data; however, because the *set_diagnosis_options* command is set to chain mode, the tool skips this diagnosis.

```
read_patterns pat_file1
set_diagnosis_options -mode chain
diagnose_failures chain_failure_file -output chain_diag.ascii -replace
diagnose_failures scan_only_failure_file -output scan_diag.ascii -csv scan_diag.csv
```

```
// Error: Cannot identify faulty chains. Please either use chain test  
// patterns or specify faulty chains.
```

Example 4

The following example sets the external test pattern source to *pattern_file1*, runs diagnosis on the *chain_foo* chain failure and the failed patterns failure file.and runs diagnosis on the *chain_foo* chain failure and the *failed cycles* failure file.

```
set_system_mode analysis  
read_patterns pattern_file1  
diagnose_failures failed_cycles -faulty_chain chain_foo FAST_TO_FALL
```

Related Topics

[display_diagnosis_report](#)
[set_diagnosis_options](#)
[set_fault_type](#)
[write_diagnosis](#)
[write_flat_model](#)
[write_failures](#)

display_diagnosis_report

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: This command can only operate on a diagnosis report file written out using the [diagnose_failures](#) command with the -Output switch.

Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the DFTVisualizer Flat Schematic window.

Usage

`display_diagnosis_report diagnose_failures_output_filename`

DFTVisualizer Menu Path: Tools > Diagnosis Report

Description

Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the DFTVisualizer Flat Schematic window.

The `display_diagnosis_report` command opens the specified diagnosis report file in File Viewer of DFTVisualizer. In the displayed report, each symptom and all suspect locations (pin and net pathnames) show up as active links. Clicking a suspect's pin.pathname link adds the gate associated with that pin to the DFTVisualizer Flat Schematic window. Clicking a suspect's net.pathname link adds all the gates connected to that net. Clicking a symptom link adds all the gates with pins in that symptom's suspect list.

Refer to "[Diagnosis Reporting](#)" in the *Tessent Diagnosis User's Manual* for a detailed description of the types of information contained in the diagnosis report.

Arguments

- ***diagnose_failures_output_filename***

A required string that specifies the pathname of a diagnosis report file written out using the `diagnose_failures` -Output command.

Examples

The following example performs scan diagnosis using the set of ATPG patterns in the pattern file `pat_file1` and the failure information in the failure file `scan_failure_file`, writing the diagnosis report in ASCII format to the file `scan_diag.ascii`. The example then opens DFTVisualizer if it is not already open and opens the diagnosis report in the File Viewer.

```
read_patterns pat_file1
diagnose_failures scan_failure_file -output scan_diag.ascii -replace
display_diagnosis_report scan_diag.ascii
```

Related Topics

[diagnose_failures](#)

[**set_diagnosis_options**](#)

display_message

Context: unspecified, all contexts

Mode: all modes

Sends a message to the specified message stream.

Usage

```
display_message string [-error | -warning | -info | -note | -screen_only] [-abort]
```

Description

Sends a message to the specified message stream.

The string can contain \n to break the message into multiple lines. Messages sent in the message stream are prefixed with an “// Error:”, “// Warning:”, “//”, “//” or “//Note.” string as appropriate, and successive lines are automatically indented.

The output of Tcl commands such as **puts** is also redirected to the screen and/or to a log file as with any other tool output.

Arguments

- ***string***

A required string that specifies the message to be sent to the message stream. The string supports the use of the newline character sequence “\n” to break the message into multiple lines. The first character of the string must be something other than a dash (“-”) character.

- **-error**

An optional literal that specifies to send a message into the error stream. This message does not interrupt the execution of the script. You can issue multiple error messages and then terminate the script using a {return -code error “string”} command. This return code causes the dofile to abort unless you have specified **set_dofile_abort** off or the -abort switch to the display_message command.

- **-warning**

An optional literal that specifies to display a warning.

- **-info**

An optional literal that specifies to display an information note. The message is prefixed by a double slash “//”.

- **-note**

An optional literal that specifies to display a note.

- **-screen_only**

An optional Boolean switch that specifies that the message string is intended for the transcript window only.

When specified, the string is not sent to the log file as it normally does and it suppresses the new line character at the end of the string. When using the -screen_only switch, you use explicit “\n” characters to force the new string to appear on a new line. You can also use the “\r” character at the beginning of the string to bring back the cursor to the beginning of the line and overwrite the existing string on the screen. Using the “\r” character is useful when you want to echo a progress status message on the screen. When you do that, make sure the new string is as long as the original string that was displayed on the line otherwise you will have left over characters from the previous string.

- **-abort**
An optional switch that forces the tool to exit.

Examples

The following example issues two error messages and then stops the invocation of the current Tcl command using a Tcl return -code error command.

```
display_message "this value is wrong" -error  
display_message "this other value is also wrong" -error  
return -code error
```

Related Topics

[register_tcl_command](#)
[unregister_tcl_command](#)

display_specification

Context: dft

Mode: setup, analysis

Displays the DftSpecification(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.

Usage

```
display_specification [id] [-design_name design_name] [-create]
```

Description

Displays the [DftSpecification](#)(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.

You can create a DftSpecification from scratch in DFTVisualizer by using the -create option, or also modify an existing DftSpecification.

When only one DftSpecification wrapper is defined in memory, you do not need to supply any arguments; the tool automatically displays the single DftSpecification wrapper.

You define or modify DftSpecifications using the Configuration window in DFTVisualizer. This window provides a graphical user interface for inserting Mentor Graphics DFT IP into existing designs. For information on using the Configuration window in DFTVisualizer, refer to the section “[How to Edit or Modify a DftSpecification](#)” in the *Tessent IJTAG User’s Manual*.

Arguments

- *id*

An optional string that specifies the DftSpecification wrapper to be displayed. As described in the [DftSpecification](#) wrapper syntax, the wrapper is identified by the two strings design_name and id. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(design_name,*), you can omit the id from the display_specification command. However, if several DftSpecification wrappers matching DftSpecification(design_name,*) exist in memory, you must specify the appropriate id when executing the display_specification command. Refer to the description of the -design_name option to understand how the value of design_name is obtained.

- -design_name *design_name*

An optional switch and string pair that specifies the design_name value used to identify the [DftSpecification](#) wrapper. The -design_name value defaults to the current design if it has been set with the [set_current_design](#) command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

When the id is specified, if exactly one wrapper matches “DftSpecification (*,<id>)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be

specified unless the -create option is specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (“*,*”)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- -create

An optional switch that disables the existence check of the DftSpecification wrapper and specifies to create and display an empty DftSpecification wrapper. If either of the id or -design_name options are not specified, their values are determined using the heuristic described in the id and -design_name options description. If the tool cannot determine the id, the default is “rtl” in the dft -rtl context and “gate” in the dft -no_rtl context. If the tool cannot determine the design_name, it is set to “?” in the created file which you can then modify in DFTVisualizer. If you do not change the “?” to a valid design name, the tool will generate an error when you attempt to process the DftSpecification.

Examples

Example 1

The following example reads a DftSpecification from a file and displays it in DFTVisualizer.

```
set_context dft -rtl  
read_config_data ./my_design.dft_spec_rtl  
display_specification
```

Example 2

The following example creates an empty DftSpecification and displays it in DFTVisualizer, allowing the user to add its content using the DFTVisualizer graphical user interface. The created DftSpecification wrapper is named “DftSpecification(mydesign,rtl)”. The design_name is taken from the name of the current design. The id defaults to “rtl” because the command was called inside the dft -rtl context.

```
set_context dft -rtl  
read_verilog mydesign.v  
set_current_design mydesign  
display_specification
```

Related Topics

[process_dft_specification](#)

dofile

Context: unspecified, all contexts

Mode: all modes

Executes the commands contained within the specified file.

Usage

dofile *filename* [-History]

Description

Executes the commands contained within the specified file.

The dofile command sequentially executes the commands contained in a specified file. This command is especially useful when you must issue a series of commands. Rather than executing each command separately, you can place them into a file in their desired order and then execute them by using the dofile command. You can also place comment lines in the file by starting the line with a double slash (//); the tool handles these lines as comments and ignores them.

The dofile command sends each command expression (in order) to the tool which in turn displays each command line from the file before executing it. If the tool encounters an error due to any command, the dofile command stops its execution, and displays an error message. You can enable the dofile command to continue regardless of errors by setting the set_dofile_abort command to Off.

The dofile command is very similar to the Tcl source command except the dofile command has the following additional usability features:

- Loads relative paths called with dofile <RelativeFilePath> relative to the position of the dofile containing the dofile command.

You must use the same syntax when using the source command as shown here:
source [file dirname [info script]]/<RelativePathToFile>. When the source command is used, the file is processed purely by the Tcl shell.
- Accepts // as comments, as well as #.
- Automatically escapes [#] and [#:#:] entries in bus names.

Arguments

- *filename*

A required string that specifies the name of the file that contains the commands that you want the tool to execute.

- -History

An optional switch that specifies for the tool to add the commands from a dofile to the command line history list. By default, the commands in a dofile are not inserted into the history list, but the dofile command itself is added to the list.

Examples

The following example executes, in order, all the commands from the file *command_file*:

dofile command_file

The *command_file* may contain any Tcl or application command available. An example of a *command_file* is as follows:

```
foreach_in_collection net [get_nets u1/*] {
    if {[sizeof_collection [get_fanins $net]] == 0} {
        delete_connections [get_fanouts $net -stop_on pin]
        delete_nets $net
    }
}
```

Related Topics

[history](#)

[set_dofile_abort](#)

Chapter 4

Command Dictionary (E - Q)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual.

Table 4-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode <u>Uncollapsed</u> Collapsed	A boldface font indicates a required argument.
[]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
<i>Italic</i>	dofile <i>filename</i>	An italic font indicates a user-supplied argument.
{ }	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	Braces enclose arguments to show grouping. Do not enter the braces.
	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort <u>ON</u> <u>OFF</u>	An underlined item indicates either the default argument or the default value of an argument.
...	add_clocks <i>off_state</i> <u>pin_list</u> ...	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Command Descriptions

This section describes commands, alphabetically from E through Q, that available in Tessent Shell. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 4-2. Commands E Through Q

Command	Description
echo	Issues a user-defined string to the transcript.
execute_cdp_test	Executes the specified test using the currently connected tester.
execute_gpib_command	Allows you to control and program GPIB equipment.
execute_tester_command	Allows you to verify and debug the wiring connection between an adaptor used for Tessent SiliconInsight and the design under test.
exit	Terminates the application tool program.
expand_compressed_patterns	Expands compressed 1-hot patterns for better internal failing flop diagnosis.
extract_icl	Checks the ICL connectivity rules between Ijtag instances, extracts the top-level ICL module and creates an SDC file that can be used for synthesis and static timing analysis (STA).
extract_sdc	Creates an SDC file that can be used for synthesis and static timing analysis (STA).
filter_collection	Filters a previously created collection, resulting in a new collection.
find_design_names	Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.
foreach_in_collection	Iterates over each element in a collection.
format_dictionary	Generates a Tcl dictionary in a human-readable format.

Table 4-2. Commands E Through Q (cont.)

Command	Description
<code>get_attribute_list</code>	Returns an alphabetically sorted Tcl list of attribute names.
<code>get_attribute_option</code>	Retrieves the current setting of an attribute's option.
<code>get_attribute_value_list</code>	Retrieves the value of an attribute on the specified design objects.
<code>get_attributed_objects</code>	Retrieves the complete collection of objects matching one of the specified object type lists and having one of the attributes found in the <i>attribute_name_list</i> specified on the object.
<code>get_auxiliary_pins</code>	Returns the input or output data or enable auxiliary pins associated to ports
<code>get_boundary_scan_port_option</code>	Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.
<code>get_cdp_test_list</code>	Returns a Tcl list of tests currently available in the CDP.
<code>get_clock_option</code>	Returns the setting of an option specified with the add_clocks command.
<code>get_clocks</code>	Returns a collection of user clocks based on filters you specify.
<code>get_common_parent_instance</code>	Returns the common ancestor of all instances, pins, or nets found in object_spec.
<code>get_config_elements</code>	Returns a collection of configuration elements or a count of configuration elements when the -count option is used.
<code>get_config_messages</code>	Returns a list of message strings attached to a configuration element.
<code>get_config_selected_nodes</code>	Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the Configuration Data window.
<code>get_config_value</code>	Returns a value associated with a configuration element based on the specified option.
<code>get_context</code>	Introspects the current context.
<code>get_current_design</code>	Returns a collection of one element that specifies the top-level design when previously set.
<code>get_current_mode</code>	Returns the current test mode, as specified by the set_current_mode command.
<code>get_current_silicon_insight_setup</code>	Returns the currently selected SiliconInsight setup as specified in the TSI setup spec.

Table 4-2. Commands E Through Q (cont.)

Command	Description
get_current_simulation_context	Returns the name of the current simulation context.
get_defaults_value	Obtains the default value of a property as specified in one of the DefaultsSpecification(company user group)/DftSpecification or DefaultsSpecification(company user group)/PatternsSpecification wrappers.
get_design_level	Returns the design level previously defined with the set_design_level command.
get_design_objects	Returns a collection of design objects associated to the object_ids supplied.
get_design_sources	Returns the pathnames or file extensions previously specified with the set_design_sources command.
get_dfm_rules	Returns a TCL list of the DFM rules contained in the layout database.
get_dft_cell	Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and desired cell function of a specified set of cells.
get_dft_info_dictionary	Creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion.
get_dft_signal	Obtains the source of a DFT signal or various aspects about the DFT signal.
get_dft_specification_requirement	Returns the requirements set using the set_dft_specification_requirements command.
get_drc_handling	Retrieves the DRC handling status for the specified <i>drc_name</i> .
get_equivalent_editable_node	Checks if a node is editable. If the node is not editable, then returns the node's editable equivalent node if it exists.
get_fanins	Returns a collection of all requested objects found in the fanin of the specified pin, net, or port object.
get_fanouts	Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.
get_fault_type	Returns the current fault model type, as previously specified by the set_fault_type command.
get_gate_pins	Returns a collection of gate_pin objects found below the current design in the flat model.

Table 4-2. Commands E Through Q (cont.)

Command	Description
<code>get_icl_extraction_options</code>	Provides access to the settings specified by the <code>set_icl_extraction_options</code> command.
<code>get_icl_fanins</code>	Returns a collection of all requested objects found in the fanin of the specified pin or port objects.
<code>get_icl_fanouts</code>	Returns a collection of all requested objects found in the fanout of the specified ICL pin, or port objects.
<code>get_icl_instances</code>	Returns a collection of all ICL instances instantiated relative to the current design that match the specified *name_patterns list or -filter expression.
<code>get_icl_modules</code>	Returns a collection of all ICL modules that match the specified name_patterns list or -filter expression.
<code>get_icl_module_parameter_list</code>	Returns the list of parameters in an ICL module.
<code>get_icl_module_parameter_value</code>	Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.
<code>get_icl_objects</code>	Returns a collection of ICL object matching the requested criteria.
<code>get_icl_pins</code>	Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified name_patterns list or -filter expression.
<code>get_icl_ports</code>	Returns a collection of all ports on a given module that match the specified name_patterns list or -filter expression.
<code>get_icl_scan_interface_list</code>	Returns the <i>names of the scan interfaces</i> in an existing ICL top module, if the ICL top module exists.
<code>get_icl_scan_interface_port_list</code>	Returns the <i>names of the ports for the specified scan interface</i> in an existing ICL top module, or the names of the ports <i>for the specified scan interface</i> that will be created in the new ICL top module during ICL extraction.
<code>get_icl_scope</code>	Introspects the current ICL scope.
<code>get_iclock_list</code>	Returns a list of ICL port and pin names on which you have issued an iClock command.
<code>get_iclock_option</code>	Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified icl_port or pin_name.
<code>get_ijtag_instances</code>	Returns a collection of instances for which there is a matched ICL module.

Table 4-2. Commands E Through Q (cont.)

Command	Description
get_ijtag_instance_option	Returns the value of the selected option for the specified IJtag instance.
get_ijtag_retargeting_options	Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not issued.
get_input_constraints	Returns the present constraint status of primary input ports or pseudo ports.
get_insertion_option	Returns the value of any insertion option that is specifiable with the set_insertion_options command.
get_insert_test_logic_option	Retrieves a prefix/infix value that will be used when logic is created by the insert_test_logic command.
get_instances	Returns a collection of all instances instantiated relative to the current design that match the specified <i>name_patterns</i> list, subject to filtering imposed by the specified options.
get_instrument_dictionary	Returns the Tcl dictionary created by the instrument processed by process_dft_specification or process_patterns_specification.
get_instrument_parent_icl_module_list	Returns the parent or grandparent module names for ICL instance module names that match the specified <i>instrument_module_name</i> .
get_iproc_argument_default	Returns the default value for the specified <i>arg_name</i> of the specified <i>proc_name</i> attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by <i>module_name</i> .
get_iproc_argument_list	Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by <i>module_name</i> .
get_iproc_body	Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by <i>module_name</i> .
get_iproc_list	Returns a Tcl list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by <i>module_name</i> .
get_layout_core_instance	Returns the current layout core instance that was set with the set_layout_core_instance command.

Table 4-2. Commands E Through Q (cont.)

Command	Description
<code>get_license_queue_timeout</code>	Returns the value specified by the previous <code>set_license_queue_timeout</code> command.
<code>get_loadboard_loopback_option</code>	Returns the settings of options specified with the <code>add_loadboard_loopback</code> pairs command.
<code>get_logfile</code>	Returns the name of and pathway to the current log file.
<code>get_logical_library_list</code>	Returns a Tcl list with all logical design library names defined with the <code>set_logical_design_libraries</code> command.
<code>get_memory_instances</code>	Returns a collection of the instances for which there is a matched TCD Memory description.
<code>get_memory_instance_option</code>	Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.
<code>get_module_parameter_list</code>	Returns the list of parameters in a module.
<code>get_module_parameter_value</code>	Returns the value of the parameter on the specified module object.
<code>get_modules</code>	Returns a collection of all modules that match the specified <code>name_patterns</code> list.
<code>get_multiprocessing_option</code>	Returns the value of a single specified variable previously set with the <code>set_multiprocessing_options</code> command.
<code>get_name_list</code>	Returns the name(s) of the specified object(s).
<code>get_nets</code>	Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified <code>name_patterns</code> list, subject to filtering imposed by the specified options.
<code>get_open_pattern_set</code>	Returns the name of the currently open pattern set.
<code>get_pattern_cycle_count</code>	Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.
<code>get_pattern_set_data</code>	Returns requested details of the internal representation of the specified pattern set.
<code>get_pattern_set_list</code>	Returns a Tcl list of all existing and closed pattern sets in the order of their creation.
<code>get_pattern_set_option</code>	Returns the value of the requested option for the specified <code>pattern_set_name</code> .

Table 4-2. Commands E Through Q (cont.)

Command	Description
get_pins	Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified name_patterns list and subject to filtering imposed by the specified options.
get_ports	Returns a collection of all hierarchical ports on a given module that match the specified name_patterns list and subjected to filtering imposed by the other options.
get_procfile_name	Returns the name of the procfile that was defined using the set_profile_name command.
get_read_verilog_option	Returns the options specified with the set_read_verilog_options command.
get_resource	Returns true when all resources in the specified resource_list are available.
get_run_synthesis_options	Returns the values for options that can be set for the run_synthesis command.
get_scan_chain_families	Gets scan chain families with certain characteristics.
get_scan_chain_option	Returns the setting of an option specified in the set_scan_chain_options command.
get_scan_elements	Returns a collection of scan elements that the tool has identified in the design.
get_scan_modes	Gets scan elements with certain characteristics.
get_scratch_directory	Returns the absolute path to the scratch directory.
get_silicon_insight_job_status	Returns the status of diagnosis jobs that are running in the background.
get_silicon_insight_option	Returns current settings of options specified in the set_silicon_insight_option command.
get_simulation_context_list	Returns the available simulation contexts in a Tcl list.
get_simulation_library_sources	Returns the pathnames or file extensions previously specified with the set_simulation_library_sources command.
get_simulation_option	Returns the values for options that can be set for the set_simulation_options command.
get_simulation_value_list	Returns a Tcl list of the simulation values on the specified gate_pin objects.

Table 4-2. Commands E Through Q (cont.)

Command	Description
get_single_attribute_value	Retrieves the value of an attribute on the specified design object.
get_single_name	Returns a string with the name of the element specified by the object_spec argument.
get_static_dft_signal_icall	Returns the iCall corresponding to the setting specified with the set_static_dft_signal_values command.
get_synchronous_clock_groups	Returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.
get_system_mode	Returns the system mode.
get_tcl_shell_option	Introspects the Tcl shell options as specified with the set_tcl_shell_options command.
get_test_end_icall_list	Returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list containing the iProc and its arguments.
get_test_point_type	Reports the current test point type as specified by the set_test_point_types command.
get_test_points	Returns a collection of all test points, or a collection of test points matching an optional Tcl list of specified test point attributes.
get_test_setup_icall_list	Returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list containing the iProc and its arguments.
get_testbench_simulation_options	Returns the value of a given option which may have been configured using the set_testbench_simulation_options command.
get_timeplate_list	Returns a list of the currently defined timeplates.
get_tool_info	Returns information about the current tool and the current run as a Tcl result.
get_tool_option	Returns certain Tesson Shell options from the set_tool_options command.
get_trace_flat_model_option	Introspects the default behavior of the trace_flat_model command.
get_transcript_style	Returns the keyword that specifies the style in which the tool transcripts commands.
get_tsdb_info	An introspection command used to extract information stored in the .tsdb_info files found in the dft_inserted_design directories of the TSDB directories.

Table 4-2. Commands E Through Q (cont.)

Command	Description
get_tsdb_list	Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.
get_tsdb_output_directory	Returns the directory used by the process_dft_specification command which is the directory that has been previously set by the set_tsdb_output_directory command.
get_validated_objects	Validates the values supplied to arguments or switches of a command created with the register_tcl_command command.
get_write_patterns_options	Returns introspection results from either port lists or vector callbacks.
get_xclock_handling	Returns the options specified with the set_xclock_handling command.
help	Lists commands and associated usage syntax.
history	Displays a list of previously executed commands.
iApply	Triggers the retargeting of all queued iRead and iWrite commands.
iCall	Calls an iProc registered against the ICL module associated with the specified <i>effective_icl_instance_path</i> .
iClock	Checks whether a controlling clock path from a valid clock source to the specified clock port exists.
iClockOverride	Models how the functional clocking has been programmed.
iComparePort	Allows you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.
identify_redundant_faults	Identifies redundant faults among current faults whose redundancy has not yet been determined.
iForcePort	Allows you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.
iMerge	Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.
import_clocks	Imports functional clocks defined during the pre-DFT design rule checking.
import_dfm	Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

Table 4-2. Commands E Through Q (cont.)

Command	Description
import_patterns_from_svf	Allows you to integrate the actions described in the SVF file into the currently open pattern set.
import_scan_mode	Imports configuration settings from tcd_scan.
index_collection	Extracts a single object at a specified index in a collection and creates a new collection containing only that object.
iNote	Inserts a note in the opened pattern set.
insert_test_logic	Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.
intercept_connection	A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.
iOverrideScanInterface	Imposes user-specified behavior on the operation of a ScanInterface.
iPDLLevel	Identifies the level and version of PDL commands.
iPrefix	Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.
iProc	Specifies a PDL procedure that can run later when referenced by the iCall command.
iProcsForModule	Specifies the ICL module for which subsequent iProc commands refer to.
iPulseClock	Allows you to start or restart a synchronous clock that previously has been forced to its off-state by means of the iForcePort command.
iRead	Adds a read operation to the command queue that will be solved by the next iApply command.
iRelease	Releases a resource previously taken by means of iTake.
iReset	Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.
iRunLoop	Creates a vector loop of a given duration.
is_collection	Indicates whether string is a collection handle.
iTakes	Takes ownership of ICL resources to prevent the retargeting software from altering their states during the processing of subsequent iApply commands or during the processing of concurrent iProcs in iMerge parallelization.

Table 4-2. Commands E Through Q (cont.)

Command	Description
iTopProc	Specifies a PDL procedure that can run later when referenced by the iCall command, when its associated ICL module is the top level module.
iUseProcNameSpace	Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.
iWrite	Adds a write operation to the command queue that will be solved by the next iApply command.
launch_sid_tester	Directs Tesson Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file.
lock_current_registration	Locks all previously registered commands and attributes preventing them from being unregistered and modified.
macrotest	Generates manufacturing test patterns for embedded logic and memories referred to as macros.
mark_display_instances	Changes the color of the specified instances in the schematic windows.
merge_cdp	Merges legacy LV flow and Tesson-Shell-based ESOE CDPs into the current Tesson Shell-based CDP.
migrate_layout	Generates an LDB directory hierarchy that contains the specified preexisting LDB.
move_config_element	Moves a configuration element from one location to another.
move_connections	Moves connections from pin or net objects to pin objects.
no_transcript	Turns off transcripting for the specified commands.
open_layout	Opens an existing Tesson Diagnosis tool-compatible Layout Database directory.
open_pattern_set	Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.
open_tsdb	Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.
open_visualizer	Opens the main DFTVisualizer window.
order_patterns	Reorders the internal test pattern set as specified.
printenv	Prints out the values of the Linux environment variables.

Table 4-2. Commands E Through Q (cont.)

Command	Description
process_dft_specification	Validates and processes the content contained in a DftSpecification wrapper.
process_patterns_specification	Validates and processes the content of the PatternsSpecification(<i>design_name, design_id, pattern_id</i>) wrapper.

echo

Context: unspecified, all contexts

Mode: all modes

Issues a user-defined string to the transcript.

Usage

`echo “string” [{> | >>} file_pathname]`

Description

Issues a user-defined string to the transcript.

The echo command issues a user-defined string to the transcript or to a pathname, if you use one of the file redirection operators.

Note

 Commands that use either the `>` or `>>` file redirection operator are first checked for correctness. Syntax errors are reported to the display prior to the command's execution. The redirection operator does not hide these errors.

Arguments

- ***string***
A required string. The string that you want echoed to the transcript. Double quotes are required if the string contains spaces or special characters.
- **`> file_pathname`**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- **`>> file_pathname`**
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example redirects output from several commands into a single output file, *my_scan_report*. The first command creates or replaces the *my_scan_report* file. The second and following commands append to the same file.

```
echo "----- scan cells -----" > my_scan_report
report_scan_cells >> my_scan_report
echo "----- scan chains -----" >> my_scan_report
report_scan_chains >> my_scan_report
```

Related Topics

[get_transcript_style](#)

[no_transcript](#)

[set_logfile_handling](#)

[set_transcript_style](#)

execute_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Executes the specified test using the currently connected tester.

Usage

```
execute_cdp_test test_name [-collect_data_type_list {data_type_list}]  
[-force_diagnose_on_failure]
```

Description

By default, this command returns pass/fail (go-nogo) results for the associated test procedure. When you specify -collect_data_type_list option, the tool collects the failure data during test execution and returns the data in a tabulated format. The tool provides the failing data with respect to the ReadVar variables originally specified in the PDL that you used to generate the pattern.

This command returns a “0” if the test passes, and a “1” if it fails, which allows you to create scripts leveraging the status.

When you are using SimDUT as your tester, the tool converts the test pattern file to simulation events, and then the Verilog simulator returns the failing information to SimDUT.

Arguments

- ***test_name***
A required string that specifies the name of the test you want to execute.
- **-collect_data_type_list *data_type_list***
An optional switch and literal that specifies the type of failure data to return. The data type options are:
 - gonogo—pass/fail results
 - variable—failing variables/registers
 - instrument—failing instruments
 - diagnosis—diagnosis datalog for failing instruments
- **-force_diagnose_on_failure**
An optional switch that forces automatic failure diagnosis if the specified test fails.

Examples

Example 1

The following example executes test1 on the DUT and returns the results for the collect_failures procedure for the test pattern associated with test1.

execute_cdp_test test1 -collect_data_type_list variable

```
// command: execute_cdp_test test1 -collect_data_type_list variable
// Test 'test1' failed.
// Variable failures and unmapped failures of pattern
// 'DLV2_1S0_ETMemory_Full.svf' :
//
// PatternSet      Variable          Pin   Expected
//                  Variable          Actual
// -----
// main            BP2_WIR.IR_STATUS2 TDO    b1
//                  b0
// -----
// main            BP2_WIR.IR_STATUS1.DONE TDO    b1
//                  b0
// -----
// main            BP2_WDR.WBP1_INST_POINTER_REG TDO  b0000000
//                  b001001
// -----
// main            BP2_WDR.WBP1_A_ADD_REG_Y TDO  b0000
//                  b1111
// -----
// main            BP2_WDR.WBP1_A_ADD_REG_X TDO  b0000
//                  b1000
// -----
// main            BP2_WDR.WBP1_GO_ID_REG10 TDO  b0
//                  b1
// -----
```

Example 2

The following example executes MemoryBist_P1 on the DUT and returns the results for the collect_failures procedure for the test pattern associated with MemoryBist_P1.

execute_cdp_test MemoryBist_P1 -collect_data_type_list {variable diagnosis}

Command Dictionary (E - Q)

execute_cdp_test

```
// command: execute_cdp_test test1 -collect_data_type_list {variable diagnosis}
// Test 'MemoryBist_P1' failed.
// Variable failures and unmapped failures of pattern 'MemoryBist_P1.stil' :
//
// PatternSet      Variable                                Pin   Expected
//                                         Actual
// -----
// run_time_prog core_inst1_top_rtl_tessent_mbist_c1_controller_inst.MBISTPG_GO(1) tdo   b1
//                                         b0
// -----
// run_time_prog core_inst1_blockA_clka_i2_mem6_interface_inst.RA_STATUS_SHADOW_REG tdo b00
//                                         b11
// -----
// run_time_prog core_inst1_blockA_clka_i2_mem6_interface_inst.GO_ID_REG tdo b000000000000
//                                         b00000000000100
// -----
// run_time_prog core_inst1_blockA_clka_i1_mem6_interface_inst.RA_STATUS_SHADOW_REG tdo b00
//                                         b11
// -----
// run_time_prog core_inst1_blockA_clka_i1_mem6_interface_inst.GO_ID_REG tdo b000000000000
//                                         b000000001000
// -----
// 
// Diagnostic Result: CDPResult/TestExecutionResult/DiagnosticResult/PatternSets(1)/
PatternsetData(0)/InstrumentResult(0)
// PatternsSpec Path: Patterns(MemoryBist_P1)/TestStep(run_time_prog)/MemoryBist/None
// Failing Controller Verilog Instance: core_inst1/top_rtl_tessent_mbist_c1_controller_inst
// Tested Controller Steps: 0-2
// Step: 2
//     Algorithm: SMARCHCHKBCIL
//     Tested Memories:
//         M12:core_inst1/blockA_clka_i1/mem6
//         M13:core_inst1/blockA_clka_i2/mem6
//     Failing Memory Verilog Instance: core_inst1/blockA_clka_i2/mem6
//     Tested Ports: 0
//     Failing Port: 0
//     Bitmap:
//     -----
//     Phase Or    Physical  Physical  Physical  Bit  Phys  Logical  Log
//     Instruction Bank    Row      Column    Pos  Exp   Address  Exp
//     -----
//     2_R1W1      0x0      0x0      0x0      3   1    0x0      1
//     3_R1W0      0x0      0x0      0x0      3   1    0x0      1
//     3_R0R0      0x0      0x0      0x0      3   0    0x0      0
//     3_R1W0      0x0      0x0      0x0      3   1    0x0      1
//     3_R0R0      0x0      0x0      0x0      3   0    0x0      0
//     5.5_FP     0x0      0x0      0x0      3   1    0x0      1
//     5.5_FP     0x0      0x0      0x1      3   1    0x1      1
//     5.5_FP     0x0      0x0      0x2      3   1    0x2      1
//     5.5_FP     0x0      0x0      0x3      3   1    0x3      1
```

Example 3

The following command returns the pass/fail results for the test pattern associated with test2.

```
execute_cdp_test test2
```

```
command: execute_cdp_test test2
Test 'test2' failed.
```

Example 4

The following script executes the MemoryBist_P1 test ten times, captures the pass/fail results, and reports the results.

```
# Execute test 10 times and capture pass/fail results
set status_array {}
for { set i 1 } { $i <= 10 } { incr i } {
    lappend status_array [ execute_cdp_test MemoryBist_P1 ]
}
# Report results after all tests complete
set i 1
foreach failflag $status_array {
    puts -nonewline "** Test $i : "
    if { $failflag } { puts "FAIL" } else { puts "PASS" }
    incr i
}
** Test 1 : PASS
** Test 2 : PASS
** Test 3 : PASS
** Test 4 : PASS
** Test 5 : FAIL
** Test 6 : PASS
** Test 7 : PASS
** Test 8 : PASS
** Test 9 : PASS
** Test 10 : PASS
SETUP>
```

Related Topics

[add_cdp_test](#)
[launch_sid_tester](#)

execute_gpib_command

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Allows you to control and program GPIB equipment.

Usage

```
execute_gpib_command command -address gpib_address [-read_result [-timeout  
read_timeout_period]]
```

Description

Note

 Using this command assumes that you have configured the equipment on the GPIB as described in General Purpose Interface Bus (GPIB) standard IEEE-488. It also assumes that the GPIB instrument supports the GPIB commands you specify.

In Tessian SiliconInsight, you generally use GPIB equipment to control power supplies and clock generators for testing purposes. However, you may also control any GPIB instrument—such as a PMU, temperature control, or digital IO—as long as it supports the GPIB standard.

You must connect the GPIB equipment to a USB-to-GPIB adaptor, which you then connect to the performance board.

Combining this command with the execute_cdp_test and add_cdp_test commands enables you to generate detailed programs for debugging and characterizing DUTs.

Arguments

- **command**

A required string that specifies a GPIB command and arguments. The GPIB command must be a valid GPIB command that is supported by the targeted GPIB instrument. Specify the command between quotes.

- **-address gpib_address**

A required switch and integer pair that specifies the address of the targeted GPIB equipment, where the *gpib_address* is an integer from 0 to 30. This switch enables execution of the GPIB command on the GPIB equipment configured to listen at the specified address.

- **-read_result**

An optional switch that directs Tessian SiliconInsight to read the results that the GPIB equipment returns after it executes the specified GPIB command. Use this switch when the GPIB command is a query command, and you expect a query result from the GPIB equipment. By default, Tessian SiliconInsight Desktop assumes that the GPIB command is a non-query command, and it does not read data from the GPIB equipment.

- **-timeout *read_timeout_period***

An optional switch and integer pair that specifies the maximum length of time (in milliseconds) to wait for the result of the specified command to appear on the GPIB before timing out. The default wait period is 2 seconds (2000 ms).

Examples

Refer to “[Examples of Controlling Instruments With GPIB](#)” in the *Tessent SiliconInsight User’s Manual for Tessent Shell* for more examples.

Example 1

The following command directs the tool to execute the SetVoltage 1.6V command on the GPIB instrument that is enabled at address 5.

```
execute_gpib_command “SetVoltage 1.6V” -address 5
```

Example 2

The following command directs the tool to execute the ReadCurrent command on the GPIB instrument that is enabled at address 12. The ReadCurrent command returns results, which Tessent SiliconInsight Desktop will read because you have specified the -read_result switch. The tool will wait no more than 500 ms for the results to appear.

```
execute_gpib_command “ReadCurrent” -address 12 -read_result -timeout 500
```

Related Topics

[add_cdp_test](#)

[execute_cdp_test](#)

execute_tester_command

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Allows you to verify and debug the wiring connection between an adaptor used for Tessonent SiliconInsight and the design under test.

Usage

```
execute_tester_command [exec_vecs adaptor_pin_list vector_list]  
[find_ir_length trst tms tdi tdo tck]
```

Description

Use the exec_vecs option to program any of the supported Tessonent SiliconInsight adaptors' pins to logic values 0 and 1. Doing this can help you debug the wiring between the adaptor and the DUT.

Use the find_ir_length option to verify that the TAP controller, if present, is wired correctly. If the command returns an error instead of the length of the instruction register, then the TAP pins are not wired correctly.

Refer to “[Debug the Wiring Connection Between the Adaptor and the DUT](#)” in the *Tessonent SiliconInsight User’s Manual for Tessonent Shell* for details.

Arguments

- *exec_vecs*
An optional literal string that specifies to execute the specified vectors at the specified logic values.
- *adaptor_pin_list*
A list that specifies one or more pin names on the adaptor that you want to control.
- *vector_list*
A list of logic values (0 or 1) for the specified pins.
- *find_ir_length*
An optional literal string that specifies to verify that the five adaptor pins that connect to TAP controller pins are wired correctly.
- *trst tms tdi tdo tck*
An ordered list of pin numbers that corresponds to the five pins that connect the adaptor to the TAP controller. You must specify the pin numbers in the order indicated by the command syntax.

Examples

The following example for a signalizerSHA40 adaptor sets pins C0 and C7 to 1 and the rest of the pins (C1, C2, C3, C4, C5, and C6) to 0:

```
execute_tester_command exec_vecs { C0 C1 C1 C3 C4 C5 C6 C7 } 10000001
```

exit

Context: unspecified, all contexts

Mode: all modes

Terminates the application tool program.

Usage

`exit [-force] [exitval]`

Description

Terminates the application tool program.

The **exit** command terminates the tool session and returns to the operating system. You should either save the current test patterns or modified design before exiting the tool.

If you are operating in interactive mode (not running a dofile) and you do not save the current test pattern set or modified design, the tool displays a warning message, and you are given the opportunity to continue the session and save the test patterns before exiting.

Arguments

- `-force`

An optional argument that forces termination of the current tool session without saving any data.

- `exitval`

An optional integer that specifies an exit code.

This option can be used to return a non-zero value when exiting the tool. By default, when there's no error, "0" is returned to the caller while non-zero usually indicates an error condition. The specific value returned can be used to differentiate the type of condition that resulted in issuing the exit command. When a tool terminates on a fatal signal whose number is *N*, the tool uses the value $128+N$ as the exit status.

Examples

The following example quits the tool without saving the current test pattern set.

```
set_system_mode analysis
create_patterns
exit -force
```

expand_compressed_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Expands compressed 1-hot patterns for better internal failing flop diagnosis.

Usage

```
expand_compressed_patterns [-CHAIN chain_name...]  
[-MAX_1hot_pattern pattern_number] [-MAP_file map_filename] [-REPlace]
```

Description

Expands compressed 1-hot patterns for better internal failing flop diagnosis.

See “[1hot Compressed Pattern Expansion](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

Arguments

- **-CHAIN *chain_name*...**
An optional switch and repeatable string that identifies the scan chain name(s) to process for 1-hot expansion.
- **-MAX_1hot_pattern *pattern_number***
An optional switch and required integer that specifies the maximum number of patterns to produce during expansion. The default value is 1000. If this limit is exceeded, the tool stops command execution and prints a warning message that the limit was exceeded.
- **-MAP_file *map_filename***
An optional switch and required string that specifies the map filename where 1-hot pattern-IDs, original pattern-IDs, and mapped channel information is saved. Use the -replace option to overwrite an existing map file.
- **-REPlace**
An optional switch for overwriting an existing file.

Related Topics

[read_patterns](#)
[set_pattern_filtering](#)
[write_patterns](#)

extract_icl

Context: all contexts

Mode: all modes

Checks the ICL connectivity rules between Ijtag instances, extracts the top-level ICL module and creates an SDC file that can be used for synthesis and static timing analysis (STA).

Usage

```
extract_icl [-write_in_tsdb {on | off | auto}] [-skip_sdc_extraction]
```

Description

Checks the ICL connectivity rules between Ijtag instances, extracts the top-level ICL module and creates an SDC file that can be used for synthesis and static timing analysis (STA).

An error is generated if an ICL module already exists for the top module (current design).

Refer to the [extract_sdc](#) command for details on SDC file creation and usage.

Using the extract_icl command is almost identical to explicitly using “set_context patterns -ijtag” followed by “set_system_mode analysis” with the following notable differences:

1. If ICL is already elaborated, using “set_system_mode analysis” in context “patterns -ijtag” will not extract a new version of the top ICL module. The tool simply transitions to analysis mode so that you can use the currently elaborated ICL to retarget PDL. When you call extract_icl, the top level ICL module is deleted and a new one is forced to be re-extracted.
2. If you change the module matching options using the set_module_matching_options command, it only has effect the next time ICL matching is done. You have to reissue the “set_context patterns -ijtag” command or the set_current_design command to redo the ICL module matching. Using the extract_icl command, the module matching is redone systematically.
3. If the ICL is already elaborated and there are input constraints added on the top level IJTAG ports, they will be automatically deleted prior to running ICL extraction. Those constraints are automatically added in a previous step if the ICL was elaborated and a setup to analysis transition was done such that the value loaded in the ICL network did not get disturbed during the state stability check.
4. If the ICL is already elaborated when the extract_icl command is called and the top ICL module contains ScanInterfaces, the name of those scan interfaces and their port list is stored before the current top level ICL module is deleted so that the original names and port list can be restored in the re-extracted top level ICL module.
5. Finally, when you use extract_icl, the context from which it was called is restored once the ICL is extracted. The “set_system_mode analysis” command leaves you in the analysis mode of the “patterns -ijtag” context.

As documented in the “[dft_inserted_designs](#)” section of the “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763, if the -write_in_tsdb switch is not specified to off, the PDL files associated to the ICL files that were checked will be collected into a single file and stored beside the extract ICL file. The extracted PDL file is also automatically sourced into memory after extraction. Each time the extracted ICL file is read in and the top ICL module elaborated using the [set_current_design](#) command, the PDL file will automatically be sourced on memory.

Arguments

- `-write_in_tsdb on | off | auto`

An optional switch and literal pair that controls whether the extracted ICL is automatically written out in the specified TSDB output directory. The TSDB output directory is specified with the [set_tsdb_output_directory](#) command and defaults to “tsdb_outdir”.

The auto value means “on” when the directory returned by [get_tsdb_output_directory](#) already exists, otherwise auto means “off.”

With the default value of “auto”, the ICL file is only written if the directory *tsdb_outdir/dft_inserted_designs/current_design_design_id.dft_inserted_design* already exists. This directory is typically created by the [process_dft_specification](#) command prior to invoking the [extract_icl](#) command as shown in [Example 1](#).

- `-skip_sdc_extraction`

An optional switch that specifies skipping SDC generation during ICL extraction.

Examples

Example 1

The following example shows the use of the [extract_icl](#) command directly following the [process_dft_specification](#) command.

```
SETUP> set_context dft -rtl
#commands to load design appear here
SETUP> set_current_design CoreA
SETUP> set_design_level physical_block
SETUP> set_system_mode analysis
ANALYSIS> create_dft_specification
ANALYSIS> process_dft_specification
ANALYSIS> extract_icl
```

Example 2

The following example shows the use of the [extract_icl](#) command as part of the “patterns -scan” context to extract the ICL network such that it can be used for test setup. For efficiency, it is recommended to extract the ICL only once after the DFT is completed, and to reuse the extracted ICL in all pattern -scan usage. ICL extraction may also not be possible after layout if the ICL modules were ungrouped during layout. The standard flow is to use the pre-layout ICL file extracted either on the RTL or netlist view for all subsequent pattern generation phases. Formal verification is typically used to verify that the functional mode, including the IJTAG network, was not affected by the layout process.

```
SETUP> set_context patterns -scan  
#commands to load design and cell library appear here  
SETUP> set_current_design  
SETUP> extract_icl  
SETUP> set_test_setup_icall myproc1
```

Related Topics

[process_dft_specification](#)

[set_system_mode](#)

[open_visualizer](#)

[set_tsdb_output_directory](#)

extract_sdc

Context: all contexts

Mode: all modes

Creates an SDC file that can be used for synthesis and static timing analysis (STA).

Usage

```
extract_sdc [-output_file output_file]
```

Description

This command creates SDC constraints for the current design and its sub-blocks for synthesis and static timing analysis (STA). Refer to “[Generating and Using SDC for Tesson Shell Embedded Test IP](#)” in the *Tesson Shell User’s Manual* for complete information on using SDC with Tesson Shell.

The extract_sdc command is run automatically as part of [extract_icl](#), therefore you would not typically run this command. If you do run extract_sdc, it requires the design’s ICL description, DFT signals and clocks. These are all automatically loaded from the TSDB using the [read_design](#) command with the -no_hdl option.

Arguments

- **-output_file *output_file***

An optional switch and file path that allows the user to explicitly specify where the SDC file is written. The default location for the SDC is in the [Tesson Shell Data Base \(TSDB\)](#) at the following location:

```
dft_inserted_designs/design_name_design_id.dft_inserted_design
```

Examples

The example runs the extract_sdc command:

INSERTION> extract_sdc

```
// Note: Changing the context to 'patterns -ijtag'.
// -----
// Begin ICL elaboration and checking.
// -----
// ICL elaboration completed, CPU time=0.05 sec.
// -----
// Warning: Input constraint 'C0' on ICL port 'bistr_clk' is removed.
// Warning: Input constraint 'C0' on ICL port 'bistr_shift_en' is removed.
// Warning: Input constraint 'C0' on ICL port 'ijtag_sel' is removed.
// Warning: Input constraint 'C0' on ICL port 'ijtag_ue' is removed.
// Warning: Input constraint 'C1' on ICL port 'ijtag_reset' is removed.
// Writing SDC file:
./tsdb_outdir/dft_inserted_designs/blockB_rtl.dft_inserted_design/
blockB.sdc
```

Related Topics

[Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions](#)

filter_collection

Context: unspecified, all contexts

Mode: all modes

Filters a previously created collection, resulting in a new collection.

Usage

```
filter_collection base_collection expression [-regexp] [-nocase]
```

Description

Filters a previously created collection, resulting in a new collection.

In many cases, the commands that create collections support a -filter option, which performs filtering as part of the collection process. Usually this is a more efficient method of filtering. However, you may want to use the filter_collection command if you are filtering a large collection multiple times with different criteria.

The filter_collection command results either in a new collection that contains a subset of the objects in the input base_collection, or in an empty collection, which indicates that the expression filtered out all elements of the input base_collection.

Arguments

- ***base_collection***

A required string that specifies the base collection to be filtered. This collection is copied to the resulting collection, and objects are removed if they are evaluated as false by the conditional expression.

- ***expression***

A required string that specifies an expression with which to filter the collection. The attributes used within the equation must exist for the object types contained in the collection. Otherwise, an error is generated. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The `=~` and `!~` matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the `-regexp` option.

- **`-regexp`**

An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **`-nocase`**

An optional switch that specifies that the pattern match is case insensitive.

Examples

Example 1

The following example creates a collection of instances of type design.

```
set var [filter_collection [get_instances *] {type == "design"}]  
          {Adder1 Adder2}
```

Example 2

The following example starts with a collection of all instances and filters the collection based on an attribute_equation of user-defined attributes.

```
set var [filter_collection [get_instances] {myType1 || myType2 == "ABC"}]
```

The above example is equivalent to this:

```
set var [get_instances -filter {myType1 || myType2 == "ABC"}]
```

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[compare_collections](#)
[copy_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

find_design_names

Context: all contexts

Mode: setup, analysis

Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.

Usage

```
find_design_names name_pattern [-LOcal | Hier]  
[-Design | -NETList | -Library | All]  
[-INstance | -Net | -Pin [INPut | OUtput | INOut | ALLIn | ALLOut] |  
[-Cell | -Module] -LIST_references]]  
[{> | >>} file_pathname]
```

Description

Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.

Arguments

- **name_pattern**
A required string, which may include asterisk (*) or question mark (?) wildcard characters.
- **-LOcal**
An optional switch that matches wildcard characters within the current hierarchy level.
- **Hier**
An optional switch that matches the regular expression across hierarchy boundaries (for example, a* matches a1/b/c). This is the default.
- **-Design**
An optional switch that matches only pathnames to objects at the topmost library cell level.
- **-NETList**
An optional switch that matches objects from the top of the design down to the topmost library cell.
- **-Library**
An optional switch that matches objects within any level of library cells.
- **All**
An optional switch that specifies matches objects at all levels of the design. This is the default.
- **INInstance**
An optional switch that matches only instance pathnames. This is the default.

- **-Net**
An optional switch that matches only net pathnames.
- **-Pin**
An optional switch that matches only pin pathnames (any pin direction). The following optional pin filters restrict which pins are matched:
 - INPut — Match only input pin pathnames.
 - OUtput — Match only output pin pathnames.
 - INOut — Match only bidirectional pin pathnames.
 - ALLIn — Match both input and bidirectional pin pathnames.
 - ALLOut — Match both output and bidirectional pin pathnames.
- **-Cell**
An optional switch that finds all library cell (model) names matching the specified regular expression.
- **-Module**
An optional switch that finds all netlist module names matching the specified regular expression.
- **-LIST_references**
An optional switch for use with -Cell or -Module that lists the pathnames of all instances in the current design that reference each cell or module matching the regular_expression.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following examples display object pathnames for various input wildcard strings, given a netlist with the following instance hierarchy:

```

/
  tiny_i
    U5
  ret_i
    intreg1_reg_0 ... intreg1_reg_31
    add_20
      U1_0 ... U1_3
    add_30
      U5 ... U12
    mul_18
      U5 ... U868
      FS
        U5 ... U33
    mul_19
      FS
        U5 ... U278
      U5 ... U181
    mul_22
      U5 ... U735
      FS
        U15 ...

```

and assuming the U5 instances all reference the following library cell:

```

model LSR2BUFA(Q, QN, S, R, G, SD, RD) (
  input(S, R, G, SD, RD) ()
  output(Q) (primitive = _buf UP1 (QT, Q);)
  output(QN) (primitive = _buf UP2 (QNT, QN);)
  intern(QT_int) (instance = LSI_LSR2 UD1 (QT_int, S, R, G, SD, RD);)
  intern(QNT_int) (instance = LSI_LSR2N UD2 (QNT_int, S, R, G, SD, RD);)
  intern(QT) (instance = LSI_NOTI UD3 (QT, QT_int);)
  intern(QNT) (instance = LSI_NOTI UD4 (QNT, QNT_int);)
)

```

Example 1

find_design_names /ret_i/add_2* -instance -design -hier

```

// Note: Matched 4 names
/ret_i/add_20/U1_0
/ret_i/add_20/U1_1
/ret_i/add_20/U1_2
/ret_i/add_20/U1_3

```

Example 2

find_design_names /ret_i/add_2* -instance -netlist -hier

```
// Note: Matched 5 names  
/ret_i/add_20  
/ret_i/add_20/U1_0  
/ret_i/add_20/U1_1  
/ret_i/add_20/U1_2  
/ret_i/add_20/U1_3
```

Finds instance add_20 under /ret_i/, and also descends the hierarchy to find all netlist instances under /ret_i/add_20/.

Example 3

```
find_design_names /ret_i/add_2* -inst -netlist -local  
  
// Note: Matched 1 names  
/ret_i/add_20
```

This example shows that -Local does not descend the hierarchy to find more matches as the previous example does.

Example 4

```
find_design_names /ret_i/add_2* -ins -design -local  
  
// Note: Matched 0 names
```

There are no instances of a library cell under /ret_i/ with instance name starting with add_2.

Example 5

```
find_design_names /ret_i/*_2? -ins -netlist -local  
  
// Note: Matched 2 names  
/ret_i/add_20  
/ret_i/mul_22
```

Found 2 instances under /ret_i/.

Note  /ret_i/gt_68_2 did not match because the “?” in the wildcard expression requires another character after the “_2”.

Example 6

```
find_design_names */U5 -inst -design -hier
```

```
// Note: Matched 7 names
/tinyos_i/U5
/ret_i/add_20/U5
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 7

find_design_names ret_i/mul*/U5 -ins -des -local

```
// Note: Matched 3 names
/ret_i/mul_18/U5
/ret_i/mul_19/U5
/ret_i/mul_22/U5
```

Example 8

find_design_names ret_i/mul*/U5 -ins -design -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 9

find_design_names ret_i/mul_18/U5* -ins -library -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/U5/UD1
/ret_i/mul_18/U5/UD2
/ret_i/mul_18/U5/UD3
/ret_i/mul_18/U5/UD4
```

Example 10

find_design_names ret_i/mul*/U5/* -pin output -design -local

```
// Note: Matched 6 names
/ret_i/mul_18/U5/Q
/ret_i/mul_18/U5/QN
/ret_i/mul_19/U5/Q
/ret_i/mul_19/U5/QN
/ret_i/mul_22/U5/Q
/ret_i/mul_22/U5/QN
```

Example 11

find_design_names * -mod -list_references

```
// Note: Found 3 netlist modules
// Note: Instances of netlist module or_gate
/o1
/o2
// Note: Instances of netlist module pme_caam_RC_CG_MOD_335839
/RC_CG_HIER_INST3921
```

Example 12**find_design_names * -cell -list**

```
// Note: Found 6 library cells
// Note: Instances of library cell HD45_SZ_MUX21X2
/mux
// Note: Instances of library cell HD45_SZ_LDLRX2
// Note: Instances of library cell HD45_SH_OR2X2
/o1/foo
/o2/foo
// Note: Instances of library cell HD45_SV_CNHLSX4
/RC_CG_HIER_INST3921/RC_CGIC_INST
// Note: Instances of library cell HD45_SH_SDFPRQX4
/data_out_reg_0
/data_out_reg_1
/data_out_reg_2
/data_out_reg_3
/data_out_reg_4
/data_out_reg_5
/shadow
/non_scan
```

foreach_in_collection

Context: unspecified, all contexts

Mode: all modes

Iterates over each element in a collection.

Usage

```
foreach_in_collection itr_var collection {body}
```

Description

Iterates over each element in a collection.

You cannot use the Tcl-supplied foreach command to iterate over collections because foreach operates on a Tcl list, and a collection is not a Tcl list but a pointer to an array of data on the C++ side. You can use the get_name_list command to convert the collection of objects into a Tcl list of object names, but you should avoid doing this if the collection is very large or if the object names are not unique, such as port names. For example, a port object is a specific port name on a specific module. Specifying just the name of the port does not indicate which module it belongs to.

You can nest the foreach_in_collection command within other control structures, including foreach_in_collection.

A collection remains in memory for as long as it is referenced by a Tcl variable. A collection is deleted once it is no longer referenced by any Tcl variable. If you make the mistake of using the Tcl command **foreach** to iterate on a collection, the command has the negative side effect of dereferencing the collection and may destroy it. Always use foreach_in_collection to iterate on the elements of a collection.

Arguments

- ***itr_var***

A required string that specifies the name of a Tcl variable. During each iteration, *itr_var* holds a string pointer to a collection of exactly one object.

- ***collection***

A required value that specifies a collection over which to iterate. It may be the content of a Tcl variable that points to a collection or the result of an introspection command that generates a collection. Note, just like the Tcl foreach command, a snapshot of the collection is taken before the first iteration. This means that even if the append_to_collection command is used to increase the size of the collection as part of the iterations, the foreach_in_collection command will not see the newly added elements. Example 3 shows how to collect new elements into a separate collection and use a Tcl while command to repeat the loop until the collection is truly empty.

- **body**

A required string that specifies a script to execute per iteration. The entire body must be enclosed within a set of braces.

Examples

Example 1

The following example connects a set of pins that has a special attribute called myAtt set to True to a net that has the same name as the pin. If not already present, the net is created. Note that the example assumes that all pins are scalar objects. Extra code would be needed to deal with bused pins and create the entire bused nets with one command. The is_bus and other built-in bus attributes are available to do this efficiently. Notice how the lindex command converts the return value of the get_attribute_value_list command into a string so that it can be used to construct the net name using \${parent}/\${leaf}.

```
foreach_in_collection pin [get_pins -filter "myAtt"] {  
    set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]  
    set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]  
    if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]} = 0} {  
        create_net ${parent}/${leaf}  
    }  
    create_connections $pin ${parent}/${leaf}  
}
```

Notice how filtering based on the value of an attribute is performed to reduce the size of the collection on which to iterate. The user could have chosen to loop on all pins and check inside the loop if the myAtt attribute is set to true, but this is very inefficient because filtering is done in C++ as opposed to looping in Tcl.

```
#inefficient use of looping. Best to filter collection before looping on  
#it as shown above.
```

```
foreach_in_collection pin [get_pins] {  
    if {[get_attribute_value_list $pin -name myAtt]} {  
        set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]  
        set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]  
        if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]} = 0} {  
            create_net ${parent}/${leaf}  
        }  
        create_connections $pin ${parent}/${leaf}  
    }  
}
```

Example 2

The following example loops on a collection of pins and deletes them. The same pin object is added twice into the collection being looped on to illustrate the fact that obsolete objects are still looped on and that a -silent switch must be used so that they are ignored by the delete_pin command. Without the -silent option, an error would be generated the second time the pin object u1/P2 was looped on.

```
set my_pins [get_pins {u1/P2 u1/P*}]
puts "my_pins are [get_name_list $my_pins]"
foreach_in_collection pin $my_pins {
    delete_pin $pin -silent
}
my pins are u1/P2 u1/P1 u1/P2 u1/P3
```

Example 3

The following example shows how to iterate on a collection of objects until no objects are left to process. The procedure called “proc_that_returns_new_objects” is called on each object and may return new objects to process. The “append_to_collection col” command cannot be used to process the new objects because the foreach_in_collection command took a snapshot of “col” before the first iteration. Instead, the new objects are collected into a collection variable called “newCol” and it is assigned back to “col” after the foreach_in_collection command is done. The loop is repeated for as long as “newCol” is not empty after the foreach_in_collection command has completed.

```
set col [<start collection>]
while {[sizeof_collection $col] > 0} {
    set newCol {}
    foreach_in_collection obj $col {
        append_to_collection newCol [proc_that_returns_new_objects $obj]
    }
    set col $newCol
}
```

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[compare_collections](#)
[copy_collection](#)
[filter_collection](#)
[get_common_parent_instance](#)
[get_modules](#)
[get_nets](#)
[index_collection](#)
[is_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

format_dictionary

Context: all including the unspecified one

Mode: all

Generates a Tcl dictionary in a human-readable format.

Usage

```
format_dictionary dictionary [-left_indent left_indent]
    [-keys_that_are_strings keys_that_are_strings]
    [-keys_that_are_lists keys_that_are_lists]
    [-list_elements_per_line {1 | all}]
    [-variable_name variable_name]
    [-banner string]
    [-preserve_collection_pointers]
```

Description

Generates a Tcl dictionary in a human-readable format. The dictionary consists of entries on separate lines with child elements indented by two white spaces relative to their parents entries.

By default, elements are considered sub-dictionaries if they contain an even number of elements. Otherwise, they are considered lists.

You can send the content of the dictionary to a file by using the *-variable_name* option.

Arguments

- ***dictionary***
A string that defines a Tcl dictionary similar to the one created using the Tcl dict command or the one returned by the [get_design_sources -file_dictionary](#) command.
- **-left_indent *left_indent***
An optional switch and value pair that controls the indentation of the dictionary, where the value is a positive integer. When specified, the tool adjusts the entire dictionary left by the specified *left_indent* value, maintaining child element indentation.
- **-keys_that_are_strings *keys_that_are_strings***
An optional switch and value pair that specifies to treat the specified leaf names of keys as strings. By default, if a string contains an even number of words, the tool treats it as a sub-dictionary. If a string contains an odd number of words, the tool considers it a list that you can control with the *-list_elements_per_line* option. When you mark a key as holding the value of a string, the string is shown inside quotes.
- **-keys_that_are_lists *keys_that_are_lists***
An optional switch and value pair that specifies to treat the specified leaf names of keys as lists. By default, if a list contains an even number of elements, the tool treats it as a sub-dictionary. When you mark a key as holding the value of a list, the list is shown inside curly

brackets ({}). You can control the number of elements to display per line with the `-list_elements_per_line` option.

- `-list_elements_per_line1 / all`

An optional switch that specifies the number of elements from a list to display on each line. By default, the entire list displays on a single line. Specify 1 to display one element per line. The tool treats the value of a key as a list when it contains an odd number of elements or it is specified by the `-keys_that_are_lists` option.

- `-variable_name variable_name`

An optional switch that specifies the name of a Tcl variable. When specified, the formatted dictionary is included in a “`set variable_name {}`” wrapper, which the tool displays in the transcript. When you specify `variable_name` with a redirection symbol (>) and a file name, the tool sends the dictionary to the specified file.

- `-banner string`

An optional switch and string pair that specifies a banner text that can be passed as a string, which can contain newlines.

- `-preserve_collection_pointers`

An optional switch that preserves the collection indexes (@<integer>). By default, a value that is a collection will be considered a list.

Examples

The following example uses the Tcl dict command to create a Tcl dictionary then shows how it looks raw versus how it looks when formated with the `format_dictionary` command.

The last invocation shows how to use the command to export the dictionary to file in a human-readable manner.

```

> dict set mydict square1 color red
> dict set mydict square1 dimensions width 3
> dict set mydict square1 dimensions height 5
> dict set mydict square1 usage "for learning"
> dict set mydict square1 partners {round triangle line point}
> puts $mydict
square1 {color red dimensions {width 3 height 5} usage {for learning}
partners {round triangle line point}}
> puts [format_dictionary $mydict]
square1 {
    color red
    dimensions {
        width 3
        height 5
    }
    usage {
        for learning
    }
    partners {
        round triangle
        line point
    }
}

> puts [format_dictionary $mydict -keys_that_are_list partners]
square1 {
    color red
    dimensions {
        width 3
        height 5
    }
    usage {
        for learning
    }
    partners {round triangle line point}
}
> puts [format_dictionary $mydict -keys_that_are_list partners
-list_elements_per_line 1]
square1 {
    color red
    dimensions {
        width 3
        height 5
    }
    usage {
        for learning
    }
    partners {
        round
        triangle
        line
        point
    }
}
> format_dictionary $mydict -keys_that_are_list partners
-list_elements_per_line 1 -variable_name mydict > mydict.txt
> cat mydict.txt
set mydict {

```

```
square1 {
    color red
    dimensions {
        width 3
        height 5
    }
    usage {
        for learning
    }
    partners {
        round
        triangle
        line
        point
    }
}
```

get_attribute_list

Context: unspecified, all contexts

Mode: all modes

Returns an alphabetically sorted Tcl list of attribute names.

Usage

Usage 1: List attributes specified on a group of objects

```
get_attribute_list obj_spec [-predefined | -user_defined] [-silent]
```

Usage 2: List attributes registered for a groups of object types

```
get_attribute_list -object_types type_list [-predefined | -user_defined] [-silent]
```

Usage 3: List all registered attributes

```
get_attribute_list [-predefined | -user_defined] [-silent]
```

Description

Returns an alphabetically sorted Tcl list of attribute names.

Usage 1: Returns the attributes set on any object found in *obj_spec*. The attributes on an object inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object are reported in the list of attributes for the instances of that module. By default, attributes that have not been set on the objects are not listed unless the attribute was registered with the -show_default option.

If an object in *obj_spec* does not exist, an error message displays unless the -silent switch is specified.

Usage 2: Returns the attributes registered for all specified object types. These do not include the inherited attributes from one object type to another such as the module attribute inherited by the associated instances.

Usage 3: Returns all registered attributes

In all three usages, only predefined attributes are returned when the -predefined option is specified. Only user-defined attributes are returned when the -user_defined option is specified.

Arguments

- *obj_spec*

A required value used in Usage 1 that specifies an object name, a Tcl list of object names, or a collection of objects.

- -object_types *type_list*

A required switch and value pair used in Usage 2 that enumerates the object types for which the attribute list is requested. The currently supported object types are module, port,

instance, pin, net, and icl_module. The list of supported object types will be augmented in successive releases and eventually all object types will be available for introspection.

- **-predefined | -user_defined**

An optional switch that constrains the returned list to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default no constraints are applied and both pre-defined and user-defined attributes are returned.

- **-silent**

An optional switch that specifies to suppress error messages if there are objects in the *object_spec* that do not exist.

Examples

Example 1

The following example lists the attributes set on three pins:

```
get_attribute_list {tiny1_3/d1 tiny1_3/d2 tiny1_3/d3}
base_name direction has_functional_source is_hard_module is_valid
leaf_name library_name master_module_name module_name name object_type
parent_instance parent_is_hard_module tie_value
```

Example 2

The following example shows the impact of using the **-show_default** switch, when registering an attribute, on the **get_attribute_list** command:

```
register_attribute -name a1 -value_type boolean -object_types pin
get_attribute_list {u1/pin1 u1/pin2} -user_defined
{ }

register_attribute -name a2 -value_type boolean -object_types pin -show_default
get_attribute_list {u1/pin1 u1/pin2} -user_defined
a2

set_attribute_value u1/pin1 -name a1
{u1/pin1}

get_attribute_list {u1/pin1 u1/pin2} -user_defined
a1 a2
```

Related Topics

[get_attribute_value_list](#)
[register_attribute](#)

[report_attributes](#)

[reset_attribute_value](#)

[set_attribute_options](#)

[set_attribute_value](#)

[unregister_attribute](#)

get_attribute_option

Context: unspecified, all contexts

Mode: all modes

Retrieves the current setting of an attribute's option.

Usage

```
get_attribute_option -name attribute_name -object_type type_name
  {[-export_during_write {auto | off | on}]
   | [-preserve_boundary_in_flat_model]
   | [-applies_to_child_instances]
   | [-display_in_gui {off| on}]
   | [-gui_marking_index marking_index]}
```

Description

Retrieves the current setting of an attribute's option.

The `get_attribute_option` command returns a string with the option settings of the specified attribute. These options can be modified after the attribute has been registered using the `set_attribute_options` command.

Arguments

- **-name** *attribute_name*

A required switch and value pair that specifies the name of the attribute whose option values are to be returned.

- **-object_type** *obj_type*

A required switch and value pair that specifies the object type whose attribute option settings are to be returned.

- **-export_during_write**

An optional switch that retrieves the current value of the `-export_during_write` option for the specified attribute name on the specified object type. For more information on the `-export_during_write` switch, see the [set_attribute_options](#) command.

- **-preserve_boundary_in_flat_model**

An optional switch that retrieves the current value of the `-preserve_boundary_in_flat_model` option for the specified attribute name on the specified object type. For more information on the `-preserve_boundary_in_flat_model` switch, see the [set_attribute_options](#) command.

- **-applies_to_child_instances**

An optional switch that retrieves the current value of `-applies_to_child_instances` for the specified attribute name on the specified object type. For more information on the `-applies_to_child_instances` switch, see the [set_attribute_options](#) command.

- [-display_in_gui](#)

An optional switch that retrieves the current value of the `-display_in_gui` option for the specified attribute name on the specified object type. For more information on the `-display_in_gui` switch, see the [set_attribute_options](#) command.

- [-gui_marking_index](#)

An optional switch that retrieves the current value of the `-gui_marking_index` for the specified attribute name on the specified object type. For more information on the `-gui_marking_index` switch, see the [set_attribute_options](#) command.

Examples

The following example retrieves the `-applies_to_child_instances` setting for the attribute named `my_power_domain` which is registered for the `instance` object type:

```
get_attribute_option -name my_power_domain -object_type instance \
    -applies_to_child_instances  
on
```

Related Topics

[get_attribute_value_list](#)
[register_attribute](#)
[report_attributes](#)
[reset_attribute_value](#)
[set_attribute_options](#)
[set_attribute_value](#)
[unregister_attribute](#)

get_attribute_value_list

Context: unspecified, all contexts

Mode: all modes

Retrieves the value of an attribute on the specified design objects.

Usage

```
get_attribute_value_list object_spec -name attribute_name [-is_specified] [-silent]
```

Description

Retrieves the value of an attribute on the specified design objects.

Like all other Tesson commands ending with an _list suffix, this command returns a list with all the values set for the attribute on each of the objects in *object_spec*. This is also the case when you pass one object. You can use lindex or a foreach loop to return the single values.

If an object in *object_spec* does not exist, or if the attribute specified by *attribute_name* does not exist on the design object, an error message displays unless the -silent switch is specified.

This command is used to introspect attributes on all object types. When *object_spec* is a Tcl list of names, the object types are queried in a specific order until one of the data models has the object by the requested name. Use “`get_attribute_value_list [get_XXX YYY]`” when you want to see the attributes of object named YYY of type XXX.

For example, the design may have a pin named u1/p1 and a net by the same name. To query the attributes of the net object, use the following:

```
get_attribute_value_list [get_nets u1/p1]
```

If you use the following:

```
get_attribute_value_list u1/p1
```

then the tool returns the attributes of the pin object because the pin object model is queried before the net object model.

The object data model types are queried in the following order:

1. module
2. instance
3. port
4. pin
5. net
6. pseudo_port

7. `gate_pin`
8. `icl_module`
9. `icl_instance`
10. `icl_instance_in_module`
11. `icl_port`
12. `icl_pin`
13. `icl_scan_interface_of_module`
14. `icl_scan_interface`
15. `icl_scan_register_in_module`
16. `icl_scan_register`
17. `icl_scan_mux_in_module`
18. `icl_scan_mux`
19. `icl_one_hot_scan_group_in_module`
20. `icl_one_hot_scan_group`
21. `icl_alias_in_module`
22. `icl_alias`
23. `config_wrapper`
24. `config_property`
25. `config_repeatable_property`
26. `config_data_wrapper`
27. `config_csv_wrapper`

Arguments

- ***object_spec***

A required value that specifies a Tcl list of one or more object names or a collection of one or more objects.

- ***-name attribute_name***

A required switch and string pair that specify the name of the attribute whose value is to be returned.

- ***-is_specified***

An optional Boolean switch that returns 1 when the attribute was explicitly set on the object otherwise a 0 is returned. When an attribute is unspecified, its value is derived from different

criteria. Instance level attributes where [get_attribute_option](#) -apply_to_child_instances returns true will inherit their value from their parent instance. If the instance does not have a parent instance, then its default value is used. Attributes that do not support inheritance always have their default value when unspecified.

- -silent

An optional switch that specifies to suppress error messages if an object in *object_spec* does not exist, or if the attribute specified by attribute_name does not exist for a given design object.

Examples

Example 1

The following example returns the value of parent_instance value for pins. The get_attribute_value_list command returns a Tcl list of values even if there is only one object in obj_spec. Use lindex as shown below to convert any one element list into a string that you can then concatenate with other strings to create object names. The 0 argument to the lindex command below takes the first (and only) element of the list.

```
get_attribute_value_list u1/u5/u8/ABC -name parent_instance  
u1/u5
```

```
get_attribute_value_list u1/DEF -name parent_instance  
{}
```

```
lindex [get_attribute_value_list u1/DEF -name parent_instance] 0
```

Example 2

The following example retrieves values for the attribute named “is_hard_module” on the listed modules.

```
get_attribute_value_list {alu mux2 rom3} -name is_hard_module  
false true false  
  
set myList {mux2 alu}  
get_attribute_value_list $myList -name is_hard_module  
true false
```

Related Topics

[get_attribute_option](#)
[register_attribute](#)
[report_attributes](#)

[reset_attribute_value](#)

[set_attribute_options](#)

[set_attribute_value](#)

[unregister_attribute](#)

get_attributed_objects

Context: unspecified, all contexts

Mode: all modes

Retrieves the complete collection of objects matching one of the specified object type lists and having one of the attributes found in the *attribute_name_list* specified on the object.

Usage

```
get_attributed_objects -attribute_names attribute_name_list -object_types type_list
```

Description

Use this command to get the complete collection of objects matching one of the specified object type list and having one of the attribute found in the *attribute_name_list* specified on the object.

This command works for any user-defined attributes and is much more efficient than getting the full collection of objects of a given type and filtering full collection based on attributes. For example, a design may have 5 millions pins but only 200 with the given attribute specified on it. Instead of filtering a 5 million object collection to find the 200 elements you care about, the commands constructs the collection directly from the attribute table.

There are two types of attributes: access attributes and storage attributes. Access attributes are attributes that reflect the value of a property from an internal data model. For example, the direction attribute of a pin is an access attribute. It reflects the direction of the pin as stored in the design database. A storage attribute on the other hand is not attached to an internal data structure. A dedicated hash table is created to store information about an object. All attributes you register with the [register_attribute](#) command are storage attribute. Most attributes not registered by you are access attribute. The `get_attributed_objects` command only works with storage attributes. If you specify the name of an access attribute with the `-attribute_names` switch, you will always get an empty collection. Use the “[report_attributes](#) `-name attribute_name`” command if you are not sure about an attribute. The command will return “Class : pre-defined vs user-defined”. When it returns “user-defined”, you can use this attribute with the `get_attributed_objects` command.

Arguments

- **-attribute_names** *attribute_name_list*

A switch-value pair used to specify a list of attribute names. The attribute names must be user-defined registered attributes for all the specified object types.

- **-object_types** *type_list*

A switch-value pair used to specify a list of object types for which you want its attributes objects.

Examples

The following example looks for all pin or port objects with the xyz_function attribute specified:

```
set attributed_objects [get_attributed_objects -attribute_names xyz_function -object_types {port pin}]
```

This example is part of “[Example 4](#)” on page 1343 in the [register_drc](#) command section. Refer to this example for the context of its usage.

get_auxiliary_pins

Context: dft, patterns

Mode: setup, analysis, insertion

Returns the input or output data or enable auxiliary pins associated to ports

Usage

```
get_auxiliary_pins port_spec [-direction {input |output}] [-enable] [-exists] [-silent]
```

Description

An introspection command used to get the auxiliary pins associated to a port. The auxiliary pin information is automatically available when the auxiliary logic is created using the [AuxiliaryInputOutputPorts](#) wrapper of the [BoundaryScan](#) or [EmbeddedBoundaryScan](#) wrapper of the [DftSpecification](#). As can be seen in [Figure 10-29](#) on page 3368 of the [AuxiliaryInputOutputPorts](#) wrapper section, the multiplexer used to inject the auxiliary output data on an output or inout pad is done on the test side of the Output JTAG multiplexer to avoid cascading two multiplexers along the functional path. This is why that when you implement boundary scan for your design, you must also implement the auxiliary data logic at the same time. If you are not using Boundary Scan in your design, you can still implement the auxiliary data logic using a [DftSpecification](#) wrapper that looks like the following:

```
read_config_data -from_string {
    DftSpecification(corea,rtl2) {
        BoundaryScan {
            BoundaryScanCellOptions {
                * : no_bscan_cell;
            }
            AuxiliaryInputOutputPorts {
                auxiliary_input_ports : gpio[11:4];
                auxiliary_output_ports : gpio[7:0];
            }
        }
    }
}
process_dft_specification
```

If you are implementing the auxiliary logic manually or with a third-party tool, you can describe the mapping between the top-level ports and the auxiliary data pins using a Tcl dictionary called “::auxiliary_data_dict” that uses the following format:

```
<port_name> {
    auxiliary_output_pin      <pin_name>
    auxiliary_output_enable_pin <pin_name>
    auxiliary_input_pin       <pin_name>
    auxiliary_input_enable_pin <pin_name>
}
... #repeat for all ports
```

You can use the Tcl “dict” command to populate this dictionary once and access the information using the `get_auxiliary_pins` command afterward. The following is an example of how to use

the Tcl “dict” command to set the auxiliary_data_input entry for the port “gpio[1]” in the dictionary:

```
dict set ::auxiliary_data_dict gpio[1] auxiliary_data_input u1/aux_in1
```

Arguments

- ***port_spec***
A required string-value pair specifying a Tcl list of port names or a collection containing port objects.
- **-direction {input |output}**
An optional switch-value pair specifying if the input or output data logic of the port is requested. Only inout ports can have both input and output auxiliary logic. When unspecified, the switch defaults to input.
- **-enable**
An optional Boolean switch used to request the enable pin, instead of the data pin, for the specified auxiliary logic direction.
- **-exists**
An optional Boolean switch used to request the existence of the auxiliary logic pin instead of the pin object itself. A 1 is returned when the requested input/output data/enable auxiliary pin exists, otherwise a 0 is returned
- **-silent**
An optional Boolean switch that suppresses the error message that is normally generated when the command is called without the -exists switch and the requested input/output data/enable auxiliary pin does not exist. Instead, an empty collection is silently returned.

Examples

The following example uses the `get_auxiliary_pins` command to get the auxiliary input and output data pins of specific ports and use their names to populate the `port_pin_name` properties found within the [EdtChannelsIn/EdtChannelsOut](#) wrappers of the [EDT DftSpecification](#) wrapper.

```

set spec [create_dft_spec -sri_sib_list {occ}]
read_config_data -in_wrapper $spec -from_string {
    Edt {
        ijtag_host_interface : Sib(edt);
        Controller(c1) {
            longest_chain_range           : 10, 50;
            scan_chain_count             : 40;
            input_channel_count          : 2;
            output_channel_count         : 1;
            connect_bscan_segments_to_lsb_chains : on;
            Connections {
                EdtChannelsIn(1) {
                }
                EdtChannelsIn(2) {
                }
                EdtChannelsOut(1) {
                    Pipelinestage {
                    }
                }
            }
        }
    }
}
set_config_value port_pin_name \
    -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(0) \
    [get_single_name [get_auxiliary_pins gpio[0] -direction input]]
set_config_value port_pin_name \
    -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(1) \
    [get_single_name [get_auxiliary_pins gpio[1] -direction input]]
set_config_value port_pin_name \
    -in $spec/Edt/Controller(c1)/Connections/EdtChannelsOut(0) \
    [get_single_name [get_auxiliary_pins gpio[2] -direction output]]

```

get_boundary_scan_port_option

Context: dft, patterns

Mode: all modes

Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.

Usage

```
get_boundary_scan_port_option {port_spec -cell_options | -pin_order_file | -pad_io_ports}
```

Description

Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.

Arguments

- ***port_spec***
A required string that specifies the name of a port or a collection containing a single port.
- **-cell_options**
A Boolean that queries the cell options that were previously specified on the given port using the “[set_boundary_scan_port_options](#) -cell_options” command.
- **-pin_order_file**
A Boolean option that queries the previously specified pin order file name using the “[set_boundary_scan_port_options](#) -pin_order_file” command.
- **-pad_io_ports**
A Boolean option that queries the previously specified collection of pad_io ports using the “[set_boundary_scan_port_options](#) -pad_io_ports” command.

Examples

The following example requests the cell_options that were previously specified on the port dac_out.

```
set_boundary_scan_port_option dac_out -cell_options analog
get_boundary_scan_port_option dac_out -cell_options
analog
```

Related Topics

[set_boundary_scan_port_options](#)
[report_boundary_scan_port_options](#)

[get_cdp_test_list](#)

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns a Tcl list of tests currently available in the CDP.

Usage

```
get_cdp_test_list
```

Arguments

None

Examples

The following example shows how you can use this command to return a Tcl list of the current tests within the CDP.

```
foreach test [get_cdp_test_list] {  
    execute_cdp_test $test  
}  
  
taptest membist1 membist2 logicbist1 logicbist2
```

Related Topics

[execute_cdp_test](#)

get_clock_option

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Returns the setting of an option specified with the [add_clocks](#) command.

Usage

```
get_clock_option object_spec
  {-type | -off_state | -pulse_always | -period [-timeplate timeplate]
   [-time_unit {ps | ns | us | ms | s}]
   | -pulse_in_capture
   | -label
   | -reference
   | -reference_inv
   | -freq_multiplier
   | -freq_divider}
   | -dft_inject_node
   | -no_dft_connections
   [-silent] [-map_pseudo_ports]
```

Description

Returns the setting of an option specified with the [add_clocks](#) command.

The [get_clock_option](#) command returns the setting of a single option type for a single defined clock. You use the [get_clocks](#) command to gather the list of defined clocks.

Note

 For more information about any of the following switches, refer to corresponding switch in the [add_clocks](#) command description.

Arguments

- *object_spec*

A required string that specifies an existing clock using the name of a port, pin, pseudo_port, label, or a collection of one port, or one pin, one pseudo_port, or one label object.

- -type

An optional switch that returns one of the following literals: sync_source, async_source, branch, or generated_source. For information about the clock types, refer to the [add_clocks](#) command description.

- **-off_state**
 An optional switch that returns 0 or 1 for synchronous clocks, and null for asynchronous clocks.
- **-pulse_always**
 An optional switch that returns a value of 1 if the clock is a pulse-always clock or 0 if it is not a pulse-always clock. A pulse-always clock is pulsed during every cycle of the pattern set.
- **-period [-timeplate *timeplate*] [-time_unit {ps | ns | us | ms | s}]**
 An optional switch that returns the clock period if specified. These two options can be used with the -period switch:
 - **-timeplate *timeplate*** — An optional switch and string pair used to identify the timeplate for determining the period from the timeplate specified in the procedure file. This option applies to, and must be specified for, synchronous clocks. When a synchronous clock is not pulsed in the specified timeplate, the time returned is three times the period of the timeplate. The “set time scale” statement in the procedure file defines the unit the tool uses for the timeplate period if you do not specify the -time_unit option.
 - **-time_unit { ps | ns | us | ms | s }** — An optional switch that specifies the time unit for the period. The time unit is not part of the returned string when using the -time_unit switch. There is no default unit for the period. The tool returns the period of synchronous clocks in the unit of the specified timeplate and the period of asynchronous clocks in the unit specified for the period in the [add_clocks](#) command.
- **-pulse_in_capture**
 An optional switch that returns a 0 or 1 to indicate whether the clock is a pulse-in-capture clock and the source is synchronous. This switch returns null when the source is asynchronous.
- **-label**
 An optional switch that returns the string value of the label name. If you did not previously specify a label, the tool supplies a unique label.
- **-reference**
 An optional switch that returns a collection with the object that the -reference option referred to when the [add_clocks](#) or [set_clock_options](#) commands was run. If the -reference switch was not specified, it returns an empty collection.
- **-reference_inv**
 An optional switch that returns a collection with the object that the -reference_inv option referred to when the [add_clocks](#) or [set_clock_options](#) commands was run. If the -reference_inv switch was not specified, it returns an empty collection.

- **-freq_multiplier**
An optional switch that returns the integer value of freq_multiplier, if specified; otherwise, it returns a 1.
- **-freq_divider**
An optional switch that returns the integer value of freq_divider, if specified; otherwise, it returns a 1.
- **-dft_inject_node**
An optional switch that returns the node object you specified with the -dft_inject_node option to the [add_clocks](#) or [set_clock_options](#) commands. If you did not specify an injection node with the -dft_inject_node switch for those commands, the tool returns a node object that is the clock node. See also the -map_pseudo_ports switch.
- **-no_dft_connections**
An optional switch that returns a 0 or 1 to indicate if the clock has DFT connections as specified with the -no_dft_connections option to the [add_clocks](#) or [set_clock_options](#) commands.
- **-silent**
An optional switch that suppresses the error message normally generated if the specified *object_spec* is not a clock, and returns a null. The -silent option has no effect on any other error reporting. You use this switch to determine if the *object_spec* is a clock. If the command “`get_clock_option object_spec -type -silent`” returns a null, you know *object_spec* is not a clock. When it returns a string, you know that *object_spec* is a clock and also its type.
- **-map_pseudo_ports**
An optional switch that specifies mapping pseudo ports to their associated pin or net objects. When using the -reference switch or the -reference_inv switch, and the returned reference or reference_inv object is a pseudo_port, the command returns the port or pin object on which the pseudo_port was created, instead of the pseudo_port object.

When -map_pseudo_ports is not specified, the tool returns the pseudo_port object. When the clock is not a pseudo_port, the switch has no effect.

If you use this switch in conjunction with the -dft_inject_node switch, the tool returns the related hierarchical pin for the pseudo_port.

Examples

Example 1

The following example adds two clocks to the clock list, reports on the clocks, and then shows several ways of using the get_clock_option command:

```
SETUP> add_clocks 0 I3 -pulse_always -label clock3
SETUP> add_clocks 1 I4 -period 50 ns -freq_multiplier 2 -label clock4
```

SETUP> report_clocks

```
User-defined Clocks (2) :
=====
Sync and Async Source Clocks
=====
-----
```

Name	Label	Off State	Constraints	Internal	Period	Freq. Multiplication
'I3'	clock3	0	Pulse always	No		
'I4'	clock4	-	Asynchronous	No	50.00ns	2

SETUP> get_clock_option I3 -label clock

clock3

SETUP> get_clock_option I4 -freq_multiplier

2

SETUP> get_clock_option I4 -period -time_unit ps

50000.0

SETUP> get_clock_option I3 -pulse_in_capture

0

SETUP> get_clock_option I4 -freq_divider

1

SETUP> get_clock_option I3 -reference

{ }

Example 2

This example demonstrates the use of the -map_pseudo_port switch. It also shows the effect when you do not use the switch.

```
add_clocks {/pll1_I1/tdr1_IPPLL1/U15/Y} -pseudo_port my_cut_pi -reference tck

// Note: Primary input 'my_cut_pi' is added at pin '/pll1_I1/tdr1_IPPLL1/U15/Y'.

add_clocks {/pll1_I1/tdr1_IPPLL1/U16/Y} -reference my_cut_pi
get_clock_option {/pll1_I1/tdr1_IPPLL1/U16/Y} -reference -map_pseudo_ports
{pll1_I1/tdr1_IPPLL1/U15/Y}
get_clock_option {/pll1_I1/tdr1_IPPLL1/U16/Y} -reference
{my_cut_pi}
```

Related Topics[add_clocks](#)

[delete_clocks](#)

[report_clocks](#)

[set_clock_options](#)

get_clocks

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Returns a collection of user clocks based on filters you specify.

Usage

```
get_clocks [name_patterns] [-match {all | pin_or_port_name | label_name}] [-regexp] [-nocase]  
[-type {sync_source | async_source | generated | branch}] [-silent] [-map_pseudo_ports]
```

Description

Returns a collection of user clocks based on filters you specify.

The `get_clocks` command returns a collection of all clock objects in your design created with `add_clocks` or a subset based on filters you specify.

Arguments

- ***name_patterns***
An optional string or Tcl list of strings that specifies one or more patterns to filter the list of returned list of clock objects.
- **-match {all | pin_or_port_name | label_name}**
An optional switch and literal pair that specifies to match the *name_patterns* with pin or port names, label names, or both (All). By default, the command matches all.
- **-regexp**
An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern.
- **-nocase**
An optional switch that directs the tool to perform case-insensitive pattern matching when looking for clocks matching the *name_patterns*.
- **-type {sync_source | async_source | generated | branch}**
An optional switch and literal pair that specifies the type of clock to return. By default, the command returns all types of clocks. For information about the clock types, refer to the “Usage” section of the [add_clocks](#) command description.
- **-silent**
An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

- **-map_pseudo_ports**

An optional switch that specifies mapping pseudo ports to their associated pins or net objects. When the clock is a pseudo_port and this switch is used, the command returns the port or pin object on which the pseudo_port was created, instead of the pseudo_port object.

When -map_pseudo_ports is not specified, the tool returns the pseudo_port object.

This switch has no effect when the clock is not a pseudo_port.

Examples

Example 1

The following example creates two sync_source clocks and then shows different ways to use the get_clocks command:

```

SETUP> add_clocks I1 -pulse_always -label clock1
SETUP> add_clocks I2 -pulse_in_capture -label clock2
SETUP> get_clocks
{I1 I2}

SETUP> get_clocks I* -match label_name
// Error: 'I*' is not a clock.
{}

SETUP> get_clocks * -match label_name
{I1 I2}

SETUP> get_clocks clock* -match label_name
{I1 I2}

SETUP> get_clocks I* -type sync_source
{I2 I1}

SETUP> get_clocks I* -type async_source
{}

SETUP> get_clocks c*2 -match label_name
{I2}

```

Example 2

This example creates a pseudo_port and then demonstrates using the -map_pseudo_port switch to return the pin information for the pseudo_port. It also shows the result when the switch is not used.

```
add_clocks {/pll1_I1/tdr1_IPLL1/U15/Y} -pseudo_port my_cut_pi -reference tck
// Note: Primary input 'my_cut_pi' is added at pin '/pll1_I1/tdr1_IPLL1/
U15/Y'

get_clocks my_cut_pi -map_pseudo_port
{pll1_I1/tdr1_IPLL1/U15/Y}

get_clocks my_cut_pi
{my_cut_pi}
```

Related Topics

[add_clocks](#)
[delete_clocks](#)
[get_clock_option](#)
[report_clocks](#)
[set_clock_options](#)

get_common_parent_instance

Context: all contexts

Mode: all modes

Returns the common ancestor of all instances, pins, or nets found in *object_spec*.

Usage

`get_common_parent_instance object_spec`

Description

Returns the common ancestor of all instances, pins, or nets found in *object_spec*.

You can use this command during dft insertion when you want to place a driving gate at the common ancestor of all its fanouts.

Arguments

- *object_spec*

A required value that specifies a Tcl list of one or more object names or a collection of one or more objects to find their common ancestor.

Examples

The following example finds the common ancestor of all pins having the attribute myAtt set to true. The name of the returned instance object is then used to instantiate a driver cell. Notice how quotes are used around the concatenation of \${parent_name} and /myGate to make sure that any white spaces that may exist in \${parent_name} (when it contains escaped identifiers) are preserved.

```
set parent [get_common_parent_instance [get_pins -filter myAtt]]
set parent_name [get_single_name $parent]
set driver [create_instance "${parent_name}/myGate" -of_module MyMod]
```

Related Topics

[create_instance](#)
[delete_instances](#)
[rename_instance](#)
[replace_instances](#)
[uniquify_instances](#)

get_config_elements

Context: unspecified, dft, patterns

Mode: all modes

Returns a collection of configuration elements or a count of configuration elements when the -count option is used.

Usage

```
get_config_elements [name_patterns] [-hierarchical] [-in_wrappers wrapper_object_spec]
    [-partition partition] [-count] [-type type] [-filter filter]
    [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of configuration elements or a count of configuration elements when the -count option is used.

When no *name_patterns* is specified, then all elements are returned unless the -in_wrappers option is specified, in which case only the elements below the specified wrappers are returned. When using one or many *name_patterns*, only the elements whose name matches at least one of the *name_patterns* are returned.

Arguments

- *name_patterns*

An optional string that specifies one name pattern or a Tcl list of name patterns to search. The name patterns are automatically decomposed into a series of anchored leaf name patterns using the / as a separator. The name identifiers are always matches ignoring case. The ids found inside the parentheses are matched taking into account case unless the -nocase option is used. The comma has special meaning inside the parentheses and is used to split the patterns and match each id independently. A leaf pattern that does not include a set of parentheses matches all elements matching the leaf name irrespective of whether it has ids or not. See the examples below for a complete illustration of the matching method.

When the -in_wrappers option is specified, the *name_patterns* are described relative to the specified wrappers. When the -hierarchical option is specified, the *name_patterns* are matched recursively against all elements below the specified wrappers.

- -hierarchical

An optional string that specifies that the *name_patterns* are matched recursively against all elements below the specified wrappers.

You should use this option with care and avoid using it inside scripts that are intended to be run continually throughout a development cycle. For example, you would not normally use the “get_pins clka -hierarchical” command to find “u1/u2/clka” in a script that is meant to run on a design as it evolves. For the same reason, you should not use the -hierarchical

option in a configuration editing script. Although a *leaf_name* may be unique today, there is no guarantee that it will remain unique in the future.

- **-in_wrappers** *wrapper_object_spec*

An option and value pair that specifies that the configuration elements are to be searched inside specific wrappers. When the **-in_wrappers** option is specified, the *name_patterns* are searched relative to the specified wrappers. The *wrapper_object_spec* can be the complete names of wrappers relative to the root of the partition or a collection containing one or many wrapper elements as returned by [get_config_elements](#) or “[get_config_value -object](#)”. When specifying the names of the wrapper, you can use the **-partition** option to specify in which partition the wrappers exist.

- **-partition** *partition_name*

An optional switch and value pair that specifies the partition from which the configuration element is to be obtained. When using the **-in_wrappers** option, the **-partition** option can only be specified if the *wrapper_object_spec* is the name of a wrapper. When the *wrapper_object_spec* is a collection containing wrapper objects, the **-partition** option cannot be specified because a specific partition is already associated to the configuration objects. All configuration data you normally interact with is in the default partition, so you typically never need to specify the **-partition** option.

The tool currently defines two partitions: the default partition in which all user specifications are stored, and the tcd partition which is a read-only partition in which the [read_core_descriptions](#) commands stores the data contained inside the Core(*module_name*) wrapper. The default partition contains the definition of [DftSpecification](#), and [DefaultsSpecification/DftSpecification](#) wrappers.

- **-count**

An optional Boolean option that specifies to return the number of elements matched as opposed to the elements themselves.

- **-type** *wrapper | data_wrapper | csv_wrapper | property*

An optional switch and value pair that specifies to limit the search to a specific configuration element type. When the **-type** option is omitted, all types are considered in the matching.

- **-filter** *attribute_equation*

An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the configuration element types being filtered. See section “[Attribute Filtering Equation Syntax](#)” for more details on filtering attribute equation format. The `=~` and `!~` matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the **-regexp** option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section “[Glob and](#)

[Regular Expression Pattern Matching Syntax](#)” for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when matching the [Ids](#) of wrapper and repeatable properties. The wrapper and property names are always matched ignoring casing.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

The following example operates on the following configuration data:

```
tmp(1) {
    ABC (Def,2) {
        prop1 : 1;
        Prop2 : 2;
    }
    abc(def,1) {
    }
    abcdef {
    }
}
```

Case 1

This matches all wrappers with a leaf name starting with abc inside wrapper tmp(1) independent of whether it has an id. The matching is case-insensitive because only the id matching is case-sensitive.

```
get_config_element abc -in tmp(1)
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 2

This matches the wrappers with the first id equal to Def. Notice that it does not match abc(def,1) as ids are matched considering casing unless the -nocase option is used.

```
get_config_elements *(Def,*) -in tmp(1)
{/tmp(1)/ABC(Def,2) }

get_config_elements *(Def,*) -in tmp(1) -nocase
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 3

This matches elements below the top-level wrapper starting with a*.

```
get_config_element */a*
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1) /tmp(1)/abcdef}
```

Case 4

This matches elements below the top-level wrapper starting with a* and having two ids.

```
get_config_element */a*(*,*)
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 5

This matches elements starting with ‘p’ anywhere inside tmp(1).

```
get_config_element p* -in tmp(1) -hierarchical
{/tmp(1)/ABC(DeF,2)/prop1 /tmp(1)/ABC(DeF,2)/Prop2}
```

Case 6

This matches elements starting with ‘p’ anywhere inside tmp(1) and returns the amount of elements that matched instead of the elements themselves.

```
get_config_element p* -in tmp(1) -hierarchical -count 2
```

Related Topics

[add_config_element](#)
[get_config_value](#)

get_config_messages

Context: unspecified, dft, patterns

Mode: all modes

Returns a list of message strings attached to a configuration element.

Usage

```
get_config_messages config_object_spec
    [-string | {-objects [-hierarchical]}]
    [-all | {[-error] [-warning] [-info]}]
```

Description

Returns a list of message strings attached to a configuration element.

By default, all message types are returned unless the -error, -warning, and/or -info options are specified. If the -objects option is specified, the command returns the configuration element if it has messages of the specified type.

Arguments

- *config_object_spec*

A required value that specifies the name of a configuration element or a collection containing one configuration element.

- -string

An optional Boolean option that specifies that the command is to return the messages formatted into strings exactly as they are shown with the [report_config_messages](#) command. The entries start with E#, W#) and I#) based on the type of the message and # is incremented from 0 independently for each type.

The use of -string is mutually exclusive with the -objects option. When either is specified, -string is assumed.

- -objects

An optional Boolean option that specifies to return a collection of configuration elements having at least one message of the specified type.

- -hierarchical

An optional Boolean option that specifies that the config_object_spec is a wrapper and that the command should return all elements below the specified wrapper having at least one configuration message for the specified types. This option is only allowed in combination with the -objects option

- **-all**
An optional Boolean option that specifies that all messages types are to be returned. The use of the -all option is mutually exclusive with the specification of the -error, -warning and -info options. When none of the four options is specified, -all is assumed.
- **-error**
An optional Boolean option that specifies that the messages returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.
- **-warning**
An optional Boolean option that specifies that the messages returned are of type warning. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning, and -info options is specified, -all is assumed.
- **-info**
An optional Boolean option that specifies that the messages returned are of type info. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

Examples

The following example adds an error message to the property rtl_extension inside DftSpecification(mydesign,rtl). The [get_config_messages](#) is then used to get the message string list on the property. The command “get_config_messages -objects -hierarchical” is also used to get a collection of all elements below DftSpecification(mydesign,rtl) with an error message.

```
set_context dft -rtl
set spec [add_config_el DftSpecification(mydesign,rtl)]
add_config_message "this is an error string \non two lines" \
    -config_element [get_config_el rtl_extension -in $spec] \
    -error -display
//   Error: /DftSpecification(mydesign,rtl)/rtl_extension
//           this is an error string
//           on two lines
get_config_messages [get_config_el rtl_extension -in $spec]
E0) this is an error string
    on two lines
get_config_messages $spec -objects -hier
{/DftSpecification(mydesign,rtl)/rtl_extension}
```

Related Topics

[add_config_message](#)
[delete_config_messages](#)
[report_config_messages](#)

get_config_selected_nodes

Context: dft, patterns

Mode: all modes

Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the Configuration Data window.

Usage

```
get_config_selected_nodes
```

Description

Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the [Configuration Data window](#).

Arguments

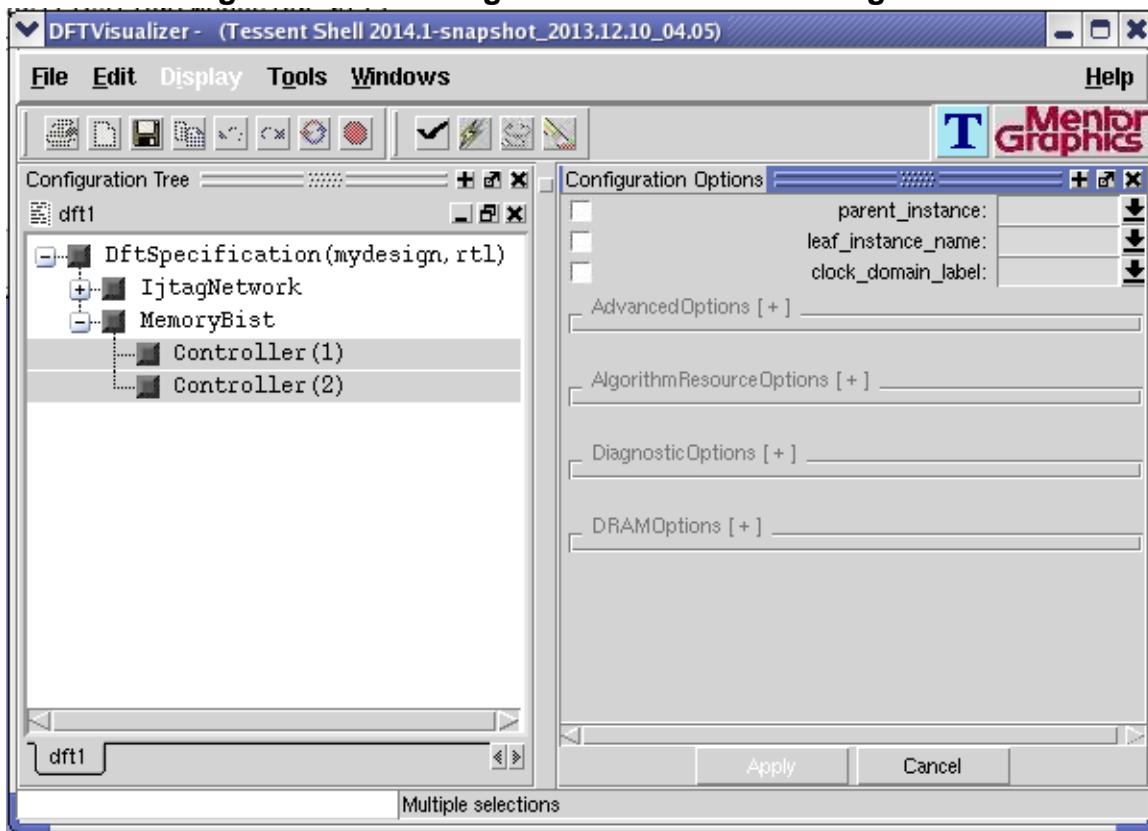
None

Examples

The following example shows the result of the command when two nodes are selected in the Configuration Data window as shown in [Figure 4-1](#). You can select more than one element by pressing the Shift or Control key while selecting the second element.

```
get_config_select_node
{/DftSpecification(mydesign,rtl)/MemoryBist/Controller(1) /DftSpecification(mydesign,rtl)/
MemoryBist/Controller(2)}
```

Figure 4-1. Selecting Two Nodes in the Configuration Tree



Related Topics

[display_specification](#)

[add_config_tab](#)

get_config_value

Context: unspecified, all contexts

Mode: all modes

Returns a value associated with a configuration element based on the specified option.

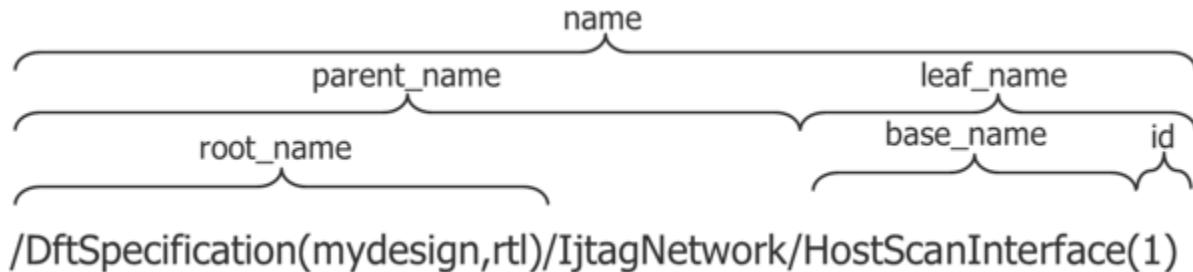
Usage

```
get_config_value config_object [-in_wrapper wrapper_object_spec]
  [-time_unit {s | ms | us | ns | ps | fs | as | as_is}]
  [-number_format {int | bin | hex | as_is}]
  [-reference_name
    | -reference_object
    | -parent_name
    | -parent_object
    | -type
    | -name
    | -id id
    | -value_id value_id
    / -is_specified
    | -has_inherited_default
    | -id_count
    | -value_count
    | -exists
    | -root_name
    | -root_object
    | -leaf_name
    | -base_name
    | -meta_name
    | -meta_object
    | -meta_id
    | -partition_name]
  [-default_value | -comment]
  [-tolower | -toupper]
  [-file_name]
  [-object]
  [-partition partition]
  [-silent]
```

Description

Returns a value associated with a configuration element based on the specified option.

[Figure 4-2](#) below illustrates some of those values.

Figure 4-2. Various Components of a Configuration Element

Arguments

- ***config_object***

A required value that specifies the name of a configuration element or a collection containing one configuration element. When ***config_object*** is a collection, the **-in_wrapper** and **-partition** options cannot be used because the object contains all of the information about the configuration element. In order to use the **-in_wrapper** option, the ***config_object*** value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)”, and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- **-in_wrapper *wrapper_object_spec***

An optional switch and value pair that is used to specify the parent wrapper in which the configuration element exists. The **wrapper_object_spec** value is the name of a wrapper or a collection containing a wrapper element. The name in ***config_object*** is relative to the specified parent wrapper.

- **-time_unit s | ms | us | ns | ps | fs | as | as_is**

An optional Boolean option that can be used when the command is returning a **Value** time property value; it is ignored otherwise. When specified or defaulted to **as_is**, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than **as_is**, the time value is converted into a real number formatted to the specified units. The unit string is also dropped.

- **-number_format int | bin | hex | as_is**

An option and value pair that can be used when the command is returning a **Value** binary property or hexadecimal value. It is ignored otherwise. When specified or defaulted to **as_is**, the value is returned unmodified with the format if it existed in the value of the configuration element. When specified to a value other than **as_is**, the binary or hexadecimal value is converted into the specified format.

- **-reference_name**

An optional Boolean option that specifies that the data requested is the name of the referenced metadata element. If the metadata is not a **Wrapper** reference property, this returns a null string.

- **-reference_object**
An optional Boolean option that specifies that the data requested is a collection containing the referenced metadata element. If the metadata is not a [Wrapper](#) reference property, this returns an empty collection.
- **-parent_name**
An optional Boolean option that specifies that the data requested is the name of the wrapper in which the configuration element exists. If the element is a root wrapper, the string / is returned.
- **-parent_object**
An optional Boolean option that specifies that the data requested is a collection containing the wrapper in which the configuration element exists. If the element is a root wrapper inside a given partition, a collection containing the partition in which the root wrapper exists is returned.
- **-type**
An optional Boolean option that specifies that the data requested is the type of the configuration element. The type is either wrapper, data_wrapper, csv_wrapper, property, repeatable_property, data_element, or csv_element. [Table 4-3](#) explains each type in more details.

Table 4-3. Types of Configuration Elements

Type	Description
wrapper	Type returned for a Wrapper used to group other wrappers and properties.
data_wrapper	Type returned for a DataWrapper used to group data_row elements.
csv_wrapper	Type returned for a CsvWrapper used to group csv_row elements.
property	Type returned for a Property .
repeatable_property	Type returned for a RepeatableProperty property.
data_row	Type returned for elements found inside a DataWrapper .
csv_row	Type returned for elements found inside a CsvWrapper .

- **-name**
An optional Boolean option that specifies that the data requested is the name of the configuration element. As shown in [Figure 4-2](#) on page 785, the name is the complete name including all hierarchical levels. If, at any level in the hierarchy, the *leaf_name* is not unique, the *leaf_name* is replaced by a positional index so as to have a unique name for each element. For example, if there are two instances of the same wrapper called “port(abc)” inside the same parent wrapper called “ports”, the name would be reported as “/Ports/Port(<2>)” assuming it is the third element inside “/ports”.

- **-id position | id_name | all**

An optional switch and value pair that specifies that the data requested is the id of a wrapper or of a repeatable property. When using -id all, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non repeatable Id declared for it. In this case the value is returned as a Tcl string which is the case for most wrappers and repeatable properties you will introspect.

When you have configuration elements with more than one Id, you can query one id at a time using either its position or when it has metadata, the id of the ID wrapper that declares the id. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on. In such a case, the id is returned as a string except when using the id of a repeatable id in which case a well-formatted Tcl list is returned.

- **-value_id position | value_name | all**

An optional switch and value pair that specifies that the data requested is the value of a property, repeatable property, or of a row inside a Data or CSV wrapper. If none of the options listed between -value_count and -reference_object inclusively are specified, “-value_id all” is assumed.

When using “-value_id all”, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the value is returned as a Tcl string which is the case for most properties you introspect.

When you have configuration elements with more than one elements, you can query one value at a time using either its position or, if it has metadata, the *id* of the Value wrapper that declares the value. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on. In such a case, the value is returned as a string except when using the id of a repeatable [Value](#) in which case a well-formatted Tcl list is returned.

- **-is_specified**

An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element is specified and 0 when it is not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and regardless of whether the singular wrapper or property is specified. The parent wrapper must exist before you can query if one of its element is specified. Unspecified elements are not included in the output file when using the [write_config_data](#) or the [report_config_data](#) commands unless the -show_unspecified option is used.

- **-has_inherited_default**

An optional Boolean option that specifies that the default is inherited. It returns true when the property is declared with the ReferenceWithInheritedDefault property. For more information on the ReferenceWithInheritedDefault property, see “[Wrapper](#)” on page 3721.

- **-id_count**
An optional Boolean option that specifies that the data requested is the number of ids specified for the wrapper of repeatable property. A value of 0 is returned for configuration objects that do not have ids like a wrapper declared without any Id wrapper or a property.
- **-value_count**
An optional Boolean option that specifies that the data requested is the number of values specified for the property, the Data or CSV wrapper row. A value of 0 is returned for configuration objects that do not have values like a wrapper. Implicit Boolean properties still report one value because the true or false value is implicit but still present.
- **-exists**
An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element exists and 0 when it does not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and whether the singular wrapper or property is specified or not. You use the **-is_specified** option to know if such elements are specified or not.
- **-root_name**
An optional Boolean option that specifies that the data requested is the name of the root wrapper in which the configuration element exists. If the element is a root wrapper, its name is returned.
- **-root_object**
An optional Boolean option that specifies that the data requested is a collection containing a root wrapper in which the configuration element exists. If the element is a root wrapper, it is returned in the collection.
- **-leaf_name**
An optional Boolean option that specifies that the data requested is the leaf name of the configuration element. As shown in [Figure 4-2](#) on page 785, the leaf name is the last level of hierarchy including the **base_name** and the **ids** when they exist.
- **-base_name**
An optional Boolean option that specifies that the data requested is the base name of the configuration element. As shown in [Figure 4-2](#) on page 785, the base name is the last level of hierarchy excluding the **ids** when they exist.
- **-meta_name**
An optional Boolean option that specifies that the data requested is the name of the [Wrapper](#) wrapper used to define the configuration element. The name is constructed assuming the metadata data was declared with no [Wrapper](#) reference property to shared declaration. The **-reference_name** option is used to get the name of the element referenced.
- **-meta_object**
An optional Boolean option that specifies that the data requested is a collection containing the [Wrapper](#) wrapper used to define the configuration element. The object is the metadata

data element as if it was declared with no [Wrapper](#) reference property to shared object. The `reference_object` option is used to get the referenced object.

- **-meta_id**

An optional Boolean option that specifies that the data requested is a unique integer corresponding to the [Wrapper](#) wrapper used to define the configuration element. The `meta_id` of an element declared with the [Wrapper](#) reference property is the `meta_id` of the referenced element.

- **-partition_name**

An optional Boolean option that specifies that the data requested is the name of the partition in which the configuration element exists. [Table 4-4](#) shows the returned value based on which partition the configuration element exists in.

- **-comment**

An optional Boolean option that specifies that the comment associated with the configuration element is to be returned as a string.

- **-default_value**

An optional switch that specifies that the data requested is the default value of a property. The command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the `default_value` is returned as a Tcl string which is the case for most properties you introspect.

You can use the `-value_id` option in combination with the `-default_value` to obtain the default value of a specific value. When requesting the default value of a single value, the return value is always a well-formatted Tcl string.

- **-tolower**

An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All upper case letters are converted to lower case.

- **-toupper**

An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All lower case letters are converted to upper case.

- **-file_name**

An optional Boolean option that specifies that the data requested is the file name from which the configuration element was read. It returns null if the configuration element was not read from a file using [read_config_data](#) but instead it was created in memory using [add_config_element](#). The `file_name` value is updated to point to the output file name as soon as it is written out using the [write_config_data](#) command.

- **-object**

An optional Boolean option that specifies that the data requested is a collection containing the configuration element just like [add_config_element](#) returns when adding an element.

- **-partition *partition_name***

An optional switch and value pair that is used to specify the partition in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications ([DftSpecification](#) and [DefaultsSpecification/DftSpecification](#)) are located. You can access the Core wrappers comprising the Tesson Core Description (TCD) file syntax in the tcd partition. Using the partition names meta: and meta:tcd allows you to access the metadata configuration elements that define the default and tcd partitions. The table below summarizes the available partition names.

Table 4-4. Available Partitions for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the Default partition is accessed which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tesson Core Description file syntax is accessed.
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

Examples

Examples

This example shows usage of the [get_config_value](#) command with a wide variety of its options exercised. You can see the syntax of the configuration data at the top and the specific configuration data being introspected just after. At the bottom of the example, you can see the metadata defining the syntax.

```
set_context dft -rtl
report_config_syntax SingularWrapper
```

```

SingularWrapper {
    WrapperWithRepeatableId(<name*>) {
        WrapperWithRepeatableId(<name*>) {
            // Same as /SingularWrapper/WrapperWithRepeatableId
        }
        time_property : <time> ; // default: 3.5ns
        integer_property : <integer> ; // default: 3
        binary_property : <binary> ; // default: 3'bo
        string_property : <string>, ... ;
        CsvWrapper(<module_name>,<design_id>) {
            <value>,...;
        }
    }
    DataWrapper_with_no_value {
        <port> ; // repeatable
    }
    DataWrapper_value {
        <port> : <period> ; // repeatable
    }
}
}

```

read_config_data ./data/cfg

report_config_data SingularWrapper

```

SingularWrapper {
    WrapperWithRepeatableId(a,b c) {
        WrapperWithRepeatableId(abc) {
            time_property : 4.5ns;
            integer_property : 6;
            binary_property : 4'b110;
            string_property : this is a string;
            CsvWrapper(modA,id1) {
                a;
                b, c;
                d, e f, g;
            }
        }
        DataWrapper_with_no_value {
            a string ;
            1;
        }
        DataWrapper_value {
            clka : 4ns;
            clkb : 6.7ms;
            clkc : 4;
        }
    }
}

```

set wrp [get_config_el WrapperWithRepeatableId(abc) -hier]

Case 1:

Get the value of a property where the type is time:

```
> puts [get_config_value time_property -in $wrp]
4.5ns
```

Case 2:

Get the time value in milliseconds with units removed:

```
> puts [get_config_value time_property -in $wrv -time_unit ms]
4.5e-06
```

Case 3:

Get the time value in nanoseconds with units removed:

```
> puts [get_config_value time_property -in $wrv -time_unit ns]
4.5
```

Case 4:

Get the value of a property where the type is integer:

```
> puts [get_config_value integer_property -in $wrv]
6
```

Case 5:

Get the value of a property where the type is binary:

```
> puts [get_config_value binary_property -in $wrv]
4'b110
```

Case 6:

Get the value of a property with a binary value converted in hexadecimal:

```
> puts [get_config_value binary_property -in $wrv -number_format hex]
4'h6
```

Case 7:

Get the value of a property with a binary value converted to integer:

```
> puts [get_config_value binary_property -in $wrv -number_format int]
6
```

Case 8:

Get the value of a property where the type is sting:

```
> puts [get_config_value string_property -in $wrv]
this is a string
```

Case 9:

Get the leaf_name of a wrapper:

```
> puts [get_config_value $wrv -leaf_name]
WrapperWithRepeatableId(abc)
```

Case 10:

Get the base_name of a wrapper:

```
> puts [get_config_value $wrp -base_name]
WrapperWithRepeatableId
```

Case 11:

Get a csv wrapper object and return the number of rows in csv wrapper:

```
> set csv [get_config_value CsvWrapper(modal,id1) -in $wrp -object]
> puts [get_config_element -in $csv -count]
3
```

Case 12:

Get the values of CSV rows:

```
> puts [get_config_value CsvWrapper(modal,id1)/<0> -in $wrp]
a
> puts [get_config_value CsvWrapper(modal,id1)/<1> -in $wrp]
b c
> puts [get_config_value CsvWrapper(modal,id1)/<2> -in $wrp]
d {e f} g
```

Case 13:

Get the number of values in a row:

```
> puts [get_config_value CsvWrapper(modal,id1)/<2> -in $wrp -value_count]
3
```

Case 14:

Get specific value of a CSV row:

```
> puts [get_config_value CsvWrapper(modal,id1)/<2> -in $wrp -value_id <1>]
e f
```

Above is equivalent to

```
> puts [lindex [get_config_value CsvWrapper(modal,id1)/<2> -in $wrp] 1]
e f
```

Case 15:

Get the number of the id for a wrapper element:

```
> puts [get_config_value -id_count \
        SingularWrapper/WrapperWithRepeatableId<0>]
2
```

Case 16:

Get the leaf_name of a wrapper element:

```
> puts [get_config_value -leaf_name $wrp]
WrapperWithRepeatableId(abc)
```

Case 17:

Get the name of a wrapper element:

```
> puts [get_config_value $wrp]
SingularWrapper/WrapperWithRepeatableId(abc)
```

Case 18:

Get the base_name of a wrapper element:

```
> puts [get_config_value -base_name $wrp]
WrapperWithRepeatableId
```

Case 19:

Get the parent_name of a wrapper element:

```
> puts [get_config_value -parent_name $wrp]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 20:

Get the parent_object of a wrapper element:

```
> puts [get_single_name [get_config_value -parent_object $wrp]]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 21:

Get the root_name of a wrapper element:

```
> puts [get_config_value -root_name $wrp]
/SingularWrapper
```

Case 22:

Get the root_object of a wrapper element:

```
> puts [get_single_name [get_config_value -root_object $wrp]]
/SingularWrapper
```

Case 23:

Get the file_name of a wrapper element:

```
> puts [get_config_value -file_name $wrp]
/home/abc/data/ex.cfg
```

Case 24:

Gets the meta_name of a wrapper element:

```
> puts [get_config_value -meta_name $wrp]  
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)
```

Case 25:

Gets the ref_name of a metadata element. Returns null when the element is not referencing another one:

```
> puts [get_conf_value \  
        SingularWrapper/WrapperWithRepeatableId/WrapperWithRepeatableId \  
        -part meta -ref_name]  
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)  
> puts [get_conf_value SingularWrapper/WrapperWithRepeatableId \  
        -part meta -ref_name]
```

Case 26:

Uses -tolower and -toupper options to convert the returned string in lower case or upper case:

```
> puts [get_config_value -base_name $wrp]  
WrapperWithRepeatableId  
> puts [get_config_value -base_name $wrp -toupper]  
WRAPPERWITHREPEATABLEID  
> puts [get_config_value -base_name $wrp -tolower]  
wrapperwithrepeatableid
```

Case 27:

Reports the metadata that defines the syntax being introspected:

report_config_data SingularWrapper -partition meta

```

Wrapper(SingularWrapper) {
    Repeatable : single;
    Wrapper(WrapperWithRepeatableId) {
        id(name*) {
            type : string;
        }
        Reference = /SingularWrapper/WrapperWithRepeatableId;
        Property(time_property) {
            Value(time) {
                type : time;
                DefaultValue : 3.5ns;
            }
        }
        Property(integer_property) {
            Value(integer) {
                type : int;
                DefaultValue : 3;
            }
        }
        Property(binary_property) {
            Value(binary) {
                type : binary;
                DefaultValue : 3'b0;
            }
        }
        Property(string_property) {
            Value(string*) {
                type : string;
                DefaultValue : "";
            }
        }
    CSVWrapper(CsvWrapper) {
        Repeatable : on;
        id(module_name) {
            Type : String;
        }
        id(design_id) {
            Type : String;
            IgnoreCase : On;
        }
    }
    DataWrapper(DataWrapper_with_no_value) {
        RowIdElement(port) {
            Type : String;
            repeatable : on;
        }
    }
    DataWrapper(DataWrapper_value) {
        RowIdElement(port) {
            Type : String;
            repeatable : on;
        }
        Value(period) {
            Type : time;
        }
    }
}
}

```

Related Topics

[set_config_value](#)

[add_config_element](#)

get_context

Context: unspecified, all contexts

Mode: all modes

Introspects the current context.

Usage

```
get_context [ -atpg | -design_identifier | -edt
            | -extraction | -failure_mapping | -full | -ijtag
            | -logic_bist | -no_rtl | -no_sub_context
            | -rtl | -scan | -scan_diagnosis | -scan_retargeting
            | -silicon_insight | -test_points ]
```

Description

Introspects the current context.

When specified without an option, this command returns dft or patterns. When specified with one of the available options, this command returns 1 if the specified sub-context is active.

Arguments

- **-atpg**
An option that returns 1 when set_context -scan is specified. When set_system_mode analysis is executed and no Logic BIST configuration is detected, the -atpg sub-context is set to 1.
- **-design_identifier**
An option that returns the specified or inferred design_identifier value.
- **-edt**
An option that returns 1 when set_context -scan is specified. When set_system_mode analysis is executed and the EDT configuration is detected, the -edt sub-context is set to 1.
- **-extraction**
An option that returns 1 when set_context -ijtag is specified. When set_current_design is executed, the extraction sub-context is set to 1 if the ICL module for the current design has not already been loaded; in this case, it will be extracted when set_system_mode analysis is executed.
- **-failure_mapping**
An option that returns 1 when set_context -failure_mapping is specified.
- **-full**
An option that returns a string that fully describes the current context, sub-context(s), and additional switches. The -full option also returns the license option when the license was

specified with the `set_context` command. This option can be used to save the current context to be restored later with the “`set_context`” command.

To save the full context, you can store it in a Tcl variable using this command:

```
set full_context [ get_context -full ]
```

To restore the full context, use this command:

```
set_context {*}$full_context
```

The {*} prefix breaks the list of arguments into separate arguments for the `set_context` command.

- **-ijtag**
An option that returns 1 when `set_context -ijtag` is specified.
- **-logic_bist**
An option that returns 1 when `set_context -scan` is specified. When `set_system_mode` analysis is executed and a Logic BIST configuration is detected, the `-logic_bist` sub-context is set to 1.
- **-no_rtl**
An option that returns 1 when `set_context -no_rtl` is either specified or inferred. It is inferred in all patterns contexts and in the dft `-scan` and `-edt` contexts.
- **-no_sub_context**
An option that returns 1 when the `set_context` was specified with no `sub_context` options such as `-scan` or `-edt`. The `-rtl/-no_rtl` options on the `set_context` command have no impact on the `get_context -no_sub_context` value.
- **-rtl**
An option that returns 1 when `set_context -rtl` is specified.
- **-scan**
An option that returns 1 when `set_context -scan` is specified.
- **-scan_diagnosis**
An option that returns 1 when `set_context -scan_diagnosis` is specified.
- **-scan_retargeting**
An option that returns 1 when `set_context -scan_retargeting` is specified.
- **-silicon_insight**
An option that returns 1 when `set_context -silicon_insight` is specified.
- **-test_points**
An option that returns 1 when `set_context -test_points` is specified.

Return Values

Returns dft or patterns when no option is specified. Returns 0 or 1 when an options is specified.

Examples

Example 1

In the following example, a script introspects the current context and displays an appropriate message.

```

if {[get_context] == "dft"} {
    if {[get_context -scan]} {
        puts "in dft scan context"
    } elseif {[get_context -edt]} {
        puts "in dft edt context"
    } else {
        if {[get_context -rtl]} {
            puts "in dft rtl context"
        } else {
            puts "in dft no_rtl context"
        }
    }
} else { ;# patterns -> get_context is patterns
    if {[get_context -scan]}{
        if {[get_context -atpg]} {
            if {[get_context -edt]} {
                puts "in patterns scan edt atpg context"
            } else {
                puts "in patterns scan atpg context"
            }
        } else { ;# logic_bist -> get_context -logic_bist is 1
            puts "in patterns scan logic_bist context"
        }
    } elseif {[get_context -ijtag]} {
        if {[get_context -extraction]} {
            puts "in ijtag icl_extraction context"
        } else {
            puts "in ijtag retargeting context"
        }
    } else { ;# scan_diagnostic
        puts "in scan_diagnostic context"
    }
}
}

```

Example 2

This demonstrates the result of the -full switch.

```

set_context dft -rtl
get_context -full

dft -rtl -design_identifier "rtl"

```

Related Topics

[report_context](#)

[set_context](#)

get_current_design

Context: all contexts

Mode: all modes

Returns a collection of one element that specifies the top-level design when previously set.

Usage

```
get_current_design [-icl | -top_config_dictionary]
```

Description

Returns a collection of one element that specifies the top-level design when previously set.

The get_current_design command returns an empty collection when the top-level design has not been previously defined using the set_current_design.

Arguments

- **-icl**
An optional switch that returns the name of an ICL module object if that object exists. The switch returns an error if there is no ICL module associated with the current design.
- **-top_config_dictionary** (only available in -rtl mode)
An optional switch that returns the name and library of the configuration of VHDL top module. You use this switch to determine if the design was elaborated using a top-level configuration or using the top module directly. When the top module was not set via a configuration, it returns a null.

The format for the returned dictionary is:

```
config_name <name> library_name <library_name>
```

Examples

Example 1

```
get_current_design
{}

read_verilog top.v
set_current_design top
get_current_design

{top}

puts [get_current_design]

@45

puts [get_single_name [get_current_design]]

top
```

```
if {[get_single_name [get_current_design]] == "top"} { puts "yes" }
yes
```

The first `get_current_design` command returns an empty collection because no current design has been set. The second `get_current_design` command returns a collection consisting of the name of the top-level design that was previously set. The first “`puts`” command returns only the pointer (@45) to a collection in memory. The second “`puts`” command shows how to use the `get_single_name` command to return the name of the module object returned by `get_current_design`.

Example 2

This example shows how to use the `-top_config_dictionary` switch to determine if the design was elaborated using a top-level configuration or using a top module directly.

```
set top_config_dictionary [get_current_design -top_config_dictionary]
if {$top_config_dictionary ne ""} {
    set top_config_name [dict get $top_config_dictionary config_name]
    set top_library_name [dict get $top_config_dictionary library_name]
} else {
    set top_module [get_current_design]
}
```

Related Topics

[set_current_design](#)
[get_name_list](#)

get_current_mode

Context: patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis,
patterns -failure_mapping

Mode: setup, analysis

Returns the current test mode, as specified by the set_current_mode command.

Usage

```
get_current_mode [-type]
```

Description

Returns the current test mode, as specified by the set_current_mode command.

If you do not include the -type switch, the command returns the name of the test mode. If you include the -type switch, this command returns the mode type.

Arguments

- -type

An optional switch that returns the mode type. Possible return values are: unwrapped, internal, external, and retargeting. For more information about these test modes, refer to the [set_current_mode](#) command description.

Examples

This example returns the test mode as “unwrapped.”

```
get_current_mode
```

```
unwrapped
```

Related Topics

[set_current_mode](#)

get_current_silicon_insight_setup

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns the currently selected SiliconInsight setup as specified in the TSI setup spec.

Usage

```
get_current_silicon_insight_setup
```

Arguments

None

Examples

```
get_current_silicon_insight_setup
```

```
sid(simdut)
```

[get_current_simulation_context](#)

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns the name of the current simulation context.

Usage

```
get_current_simulation_context
```

Description

Returns the name of the current simulation context.

The `get_current_simulation_context` command returns the current simulation context. For a list of predefined simulation contexts, refer to [Table 6-7](#) on page 2023.

Arguments

None.

Examples

The following example shows how to set and return the current simulation context:

```
SETUP> get_current_simulation_context
stable_after_setup

SETUP> set_current_simulation_context stable_load_unload
SETUP> get_current_simulation_context

stable_load_unload
```

Related Topics

[add_simulation_context](#)
[copy_simulation_context](#)
[delete_simulation_contexts](#)
[get_simulation_context_list](#)
[report_simulation_contexts](#)
[set_current_simulation_context](#)

get_defaults_value

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Obtains the default value of a property as specified in one of the DefaultsSpecification(company | user | group)/DftSpecification or DefaultsSpecification(company | user | group)/PatternsSpecification wrappers.

Usage

```
get_defaults_value property_path_name
    [-policy policy]
    [-time_unit time_unit] [-number_format format]
    [-is_specified] [-tolower] [-toupper]
```

Description

Obtains the default value of a property as specified in one of the DefaultsSpecification(company | user | group)/DftSpecification or DefaultsSpecification(company | user | group)/PatternsSpecification wrappers.

When the -policy option is not specified, it returns the value specified in the [DefaultsSpecification\(user\)](#) policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

When automatically resolving the priority, the default value of the property is accessed.

Arguments

- ***property_path_name***

A required string that defines the path to a property without the “DefaultsSpecification()” prefix. For the DftSpecification, this path starts with “DftSpecification/”. For the PatternsSpecification, this path starts with “PatternsSpecification/”.

- **-policy *policy***

An optional switch and value pair that defines from which DefaultsSpecification wrapper the default value is requested. If not specified, the command returns the value specified in the user policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

- **-time_unit *time_unit***

An optional switch and value pair that is only relevant when obtaining the value of a property of type time. This option allows you to request the time expressed in the units you want. For more information see the description of the [get_config_value](#) -time_unit option.

- **-number_format *format***

An optional switch and value pair that is only relevant when obtaining the value of a property of type binary. This option allows you to request the binary number expressed in the format you want. For more information see the description of the [get_config_value](#) -number_format option.

- **-is_specified**

An optional Boolean option that specifies that you are not requesting the actual value of the property but instead requesting status on whether the property is specified or not in the given policy. If the -policy option is not specified, the tool returns 1 if it is specified in any of the policy wrappers. If the -policy option is not specified and the tool returns 0, then you know you are getting the tool default value.

- **-tolower**

An optional switch and value pair that is only relevant when obtaining the value of a property of type string. This option allows you to request the string with all upper-case letters converted to lower case.

- **-toupper**

An optional switch and value pair that is only relevant when obtaining the value of a property of type string. This option allows you to request the string with all lower-case letters converted to upper case.

Examples

The following example queries the default value of a property and then checks which policy specified it:

```
set p DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in?
get_defaults_value $p
my_scan_in

foreach policy {user group company} {
    if {[get_defaults_value $p -policy $policy -is_specified]} {
        break
    }
}
puts "$p is specified by policy $policy."
'DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in' is specified by policy
'group'.
```

Related Topics

[set_defaults_value](#)

[create_dft_specification](#)

[get_design_level](#)

Context: dft

Mode: setup, analysis, insertion

Returns the design level previously defined with the set_design_level command.

Usage

`get_design_level`

Description

Returns the design level previously defined with the set_design_level command.

When a new design is elaborated, the level goes back to being unspecified. In this case, the get_design_level command returns a null string. When using this command, “top” is treated as a synonym to “chip”.

Arguments

None

Examples

The following example queries the level of the current design and stores the result in a variable.

```
set design_level [get_design_level]
```

Related Topics

[set_design_level](#)

[set_current_design](#)

get_design_objects

Context: All

Mode: All

Returns a collection of design objects associated to the object_ids supplied.

Usage

```
get_design_objects {-from_ids object_ids | -from_objects from_objects}  
[-map_pseudo_ports]
```

Description

This command converts a list of *object_ids* or objects to a collection of objects. Each design object type (module, instance, port, pin, net, and pseudo_port) has a unique *object_id*, as described in the “[Hierarchical Design Data Model](#)” section.

The value of these attributes consists of a string followed by a colon and a unique integer per object type. The string is the name of the object type. For example, the object id of a module is module:integer. You use this unique string to construct a dictionary for objects, using the *object_id* as the key. You use the *object_ids* with the “get_design_objects -from_ids” command to get back to the objects. Using the *object_id* rather than the name of the object is more efficient and more robust because it is much shorter and does not change during synthesis.

The object names found inside a generate block in RTL change during synthesis. For example, the instance u1 inside generate block b1 has a name b1.u1 in RTL but “\b1.u1” after quick synthesis. When you build a dictionary in analysis mode with the synthesized name as a key, you will be unable to find the object with that name in insertion mode because the RTL view is restored in insertion mode. When you store the *object_id*, you can find the object in either the synthesized view or the RTL view. See the example below for an illustration of this behavior.

Arguments

- **-from_ids *object_ids***

A required switch-value pair that is used to supply a Tcl list of *object_ids*. The *object_ids* of the objects are obtained using the “[get_attribute_value_list -name *object_id*](#)” command. An *object_id* can only be used to get back the objects for as long as the design remains elaborated in memory. There is no guarantee that an *object_id* for a given object remains the same the next time the design is elaborated. It is exactly like the gate_id of a gate_pin object that is only valid while the flat model is in memory. Never save a dictionary to disk containing *object_ids* and expect the *object_ids* to have meaning when you reload the dictionary at a later time. The *object_ids* are only stored in memory and reused while the design remains in memory as shown in the example.

- **-from_objects *from_objects***

A required switch-value pair that is used to supply a Tcl list of objects. The ***from_objects*** may contain ports, pseudo_ports, pins, nets and gate_pins. A port, pin or net is passed through to the result collection.

- **-map_pseudo_ports**

An optional switch that specifies a pseudo_port is mapped to the corresponding pins or nets, otherwise it is just passed through to the result collection. A gate_pin is mapped to the corresponding port, pseudo_port, pin or net. If it maps to a pseudo_port and “-map_pseudo_ports” is specified that pseudo_port is further mapped to the corresponding pins or nets as described above.

Examples

Example 1

The following example shows that the name of an object changes with synthesis but the ***object_id*** does not. The ***object_id*** stored in analysis from the synthesized view is valid to get the RTL object in insertion mode.

```

SETUP> set design_mods [get_modules -of_type design]
SETUP> set_attribute_value $design_mods -name synthesize_before_analysis
SETUP> set_system_mode analysis

// -----
// Begin RTL synthesis.
// -----
// Module 'FF' synthesized.
// Module 'elt' synthesized.
// Module 'subBlock1' synthesized.
// Module 'subBlock2' synthesized.
// RTL synthesis completed, synthesized modules=4, Time=1.83 secs.
// -----


ANALYSIS> set subBlock1_instances [get_instance -of_module subBlock1]ANALYSIS> set
subBlock1_instance1 [index_collection $subBlock1_instances 1]

ANALYSIS> set name [get_single_name $subBlock1_instance1]
ANALYSIS> set object_id [get_attribute_value_list $subBlock1_instance1 \
-name object_id]

puts [get_single_name [get_instance [list $name]]] option1.subBlock2_I2 /subBlock1
puts [get_single_name [get_design_objects -from_ids $object_id]] \
option1.subBlock2_I2 /subBlock1

ANALYSIS> set_system_mode insertion

# The object name changed when restoring the RTL view but the
# object_id remained the same

INSERTION> puts [get_single_name [get_instance [list $name]]]

// Warning: No result.

```

```
INSERTION> puts [get_single_name [get_design_objects -from_ids $object_id]] \\\noption1.subBlock2_I2/subBlock1
```

Example 2

The following example shows how to map pseudo ports and/or gate_pins to port, pin and net objects. Any pin, port or net objects found in \$col is passed through unchanged. Any gate_pin object is mapped to its associated port, pin, net objects. If the gate_pin objects is inside a cell and has no associated port, pin, or net objects, it is silently deleted. Finally, any pseudo_port objects found in \$col is mapped to its associated port, pin, net object because the -map_pseudo_ports option is used. When the switch is not used, the pseudo_port objects are passed through unchanged.

```
set design_object [get_design_objects -from_objects Scol1 -map_pseudo_ports]
```

Related Topics

[get_attribute_value_list](#)

get_design_sources

Context: unspecified, all contexts

Mode: setup, analysis, insertion

Returns the pathnames or file extensions previously specified with the `set_design_sources` command.

Usage

```
get_design_sources [-format verilog] [-path_list] [-extension_list] [-file_dictionary]
```

```
get_design_sources [-format icl [-search_design_load_path]]
```

Description

Returns the pathnames or file extensions previously specified with the `set_design_sources` command.

By default, the `get_design_sources` command returns a list of paired values that specify the pathnames to all design sources. The first item in the pair is `-v` or `-y` to indicate whether the pathname is a file or a directory, respectively. This is useful when you want to add additional design sources because any invocation of the `set_design_sources` command overwrites previous invocations.

Arguments

- `-format {verilog | icl}`

An optional switch that allows you to choose Verilog modules or ICL modules. By default, the command specifies Verilog modules.

- `-path_list | -extension_list | -file_dictionary | -search_design_load_path`

A 1-of-n choice of optional switches that return options specified with the `set_design_sources` command:

`-path_list` — Returns a list of pathnames. This is the default.

`-extension_list` — Returns a list of file extensions.

`-file_dictionary` — Returns a Tcl dictionary that contains information about the design files that the tool has read and written. This can be useful, for example, if you need to create a script for a downstream tool that needs to read the correct design files with the correct options in the correct sequence. For a description of the file dictionary format, see the description of the `<design_name>.design_source_dictionary` component in the `dft_inserted_designs` section of the TSDB chapter.

`-search_design_load_path` — Returns a Boolean value (on or off) that indicates whether the `set_design_sources -search_design_load_path` option is enabled. You can use this switch only when you also use the `-format` switch.

Examples

This example clears previous set_design_sources commands before setting new design sources, and then returns the list of pathnames and the list of extensions:

```
SETUP> set_design_sources -clear
SETUP> set_design_sources
// No -v/-y files/directories specified.

SETUP> set_design_sources -v MODB.v MODC.v -y /u/joe/testcases -extensions .v .vhf
SETUP> get_design_sources -path_list -v MODB.v -v MODC.v -y /u/joe/testcases
SETUP> get_design_sources -extension_list .v .v.gz .vhf .vhf.gz
```

Related Topics

[report_design_sources](#)
[set_design_sources](#)

get_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Returns a TCL list of the DFM rules contained in the layout database.

Usage

```
get_dfm_rules [-header | -col columns...]
```

Description

Returns a TCL list of the DFM rules contained in the layout database.

By default, for each rule the tool returns the id, rule name, layer, and type.

For more information about DFM rules, see “[Diagnosis for Design for Manufacturability Analysis](#).”

Arguments

- **-header**
An optional switch that returns a list of available column names.
- **-col *columns...***
An optional switch that specifies the column names you want to use for the DFM rule information in the list.

Examples

Example 1

The following example sets the available column names of the DFM report table into a TCL variable and displays the column names.

```
open_layout MyLayoutDB
set columns [ get_dfm_rules -header ]
puts "columns: $columns"

columns: id rule layer type
```

Example 2

The following example creates a custom report by inspecting the DFM rules:

```
foreach { id rule_name layer_name type_name } [ get_dfm_rules ] {
    puts "$id $rule_name $layer_name $type_name"
}
```

```

1 minimum_space_4_line_route1 route_1 BRIDGE
2 metal_cross_edge_route1 route_1 OPEN
3 low_density_min_space_route2 route_2 BRIDGE
4 half_comb_route3 route_2 OPEN
5 metal_cross_edge_route1 route_3 OPEN
6 corner_to_interior_metal4 route_4 BRIDGE
7 divergent_line_with_jog_metal5 route_5 BRIDGE
8 metal_cross_edge_route1 route_5 OPEN
9 route_5_barbell route_5 OPEN
10 route_2_between_vias route_2 OPEN

```

Example 3

The following example customizes the columns for the report:

```

foreach { r t } [ get_dfm_rules -col rule type ] {
    puts "$r $t"
}

minimum_space_4_line_route1  BRIDGE
metal_cross_edge_route1  OPEN
low_density_min_space_route2  BRIDGE
half_comb_route3  OPEN
metal_cross_edge_route1  OPEN
corner_to_interior_metal4  BRIDGE
divergent_line_with_jog_metal5  BRIDGE
metal_cross_edge_route1  OPEN
route_5_barbell  OPEN
route_2_between_vias  OPEN

```

Related Topics

[report_dfm_rules](#)

get_dft_cell

Context: dft

Mode: all modes

Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and desired cell function of a specified set of cells.

Usage

```
get_dft_cell -available_cell_selection_names | -default_cell_selection_name | function_name
[-with_asynch_enable] [-with_asynch_disable] [-with_set] [-with_reset] [-length length]
[-cell_selection_name dft_cell_selection_name] [-inputs number]
[-with_inverted_inputs number_inverted_inputs] [-silent]
```

Description

Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and desired cell function of a specified set of cells.

If only one set (one cell_selection) exists, or the “set_cell_library_options -default_cell_selection_name” command was used to set the default cell selection, then the -cell_selection_name switch is optional for retrieving a cell of a given function. Otherwise, -cell_selection_name must be used to identify a specific cell selection.

This command is very useful in enabling you to write a technology-independent command. After you have retrieved the cell module using the get_dft_cell command, you can use the function attribute on the ports of the module to find the ports you want to connect to.

For more information on Tessent cell library attributes, see the [Tessent Cell Library Manual](#). More specifically, refer to the dft_cell_selection wrapper description described in the [Cell Selection](#) section as well as the function attribute described in [Pin Attributes](#).

You can see a list of the available DFT cell functions by using the “help get_dft_cell” command. Each of the listed functions corresponds one for one with the elements documented in the dft_cell_selection wrapper found in the [Cell Selection](#) section of the [Tessent Cell Library Manual](#). Many basic functions are learned and automatically assigned to a dft_cell_selection wrapper named default when your library does not contain a dft_cell_selection wrapper.

Note

 If your Tessent cell library contains only ATPG model information but you have a cell.lib file in the original LogicVision format, you can load it natively using the [read_cell_library](#) command and the dft_cell_selection wrapper will be populated.

Arguments

- `-available_cell_selection_names`

An optional switch that specifies for the tool to return a Tcl list of names of the currently available DFT cell selections. If none are parsed and one is learned by default, it will return a list with the single element “default”. If there is no cell selection parsed and no default is created (for example, if no library is loaded) it will return null.

- `-default_cell_selection_name`

An optional switch that will return null unless there is only one cell selection or the [set_cell_library_options -default_dft_cell_selection_name](#) command was used to define a cell selection as the default.

- `function_name`

A optional string that specifies the DFT cell function from the specified or default `dft_cell_selection` to return. If the specified function does not exist in the specified `dft_cell_selection`, the tool issues an error and lists the `dft_cell_selection` names that contains the requested `function_name`.

- `-with_asynch_enable`

An optional switch that applies to `clock_gating_and` as well as `clock_gating_or` and specifies to return a cell with an asynchronous input to enable clocking at the cell output.

- `-with_asynch_disable`

An optional switch that applies to `clock_gating_and` as well as `clock_gating_or` and specifies to return a cell with an asynchronous input to disable clocking at the cell output.

- `-with_set`

An optional switch that applies to `active_high_latch`, `active_low_latch`, `posedge_dff`, `negedge_dff`, `posedge_scan_cell`, `negedge_scan_cell`, `posedge_synchronizer_cell` or `negedge_synchronizer_cell` and specifies to return a cell with an asynchronous input to set the cell’s state to “1” (high). For all types except `posedge_synchronizer_cell` and `negedge_synchronizer_cell`, if a cell with exactly what is requested does not exist, null will be returned. However, for the synchronizer cells, a cell will be returned if it has all the features requested, and can be tied off outside to produce the requested cell. For example, if a synchronizer_cell with no asynchs is requested (no `-with_set` or `-with_reset` switches) and none exists, but a sync cell with one or more asynchs does exist, it will be returned.

Similarly, if a sync_cell `-with_set` is requested, and none with only a set exists, but one with both asynchs does exist, it will be returned.

- `-with_reset`

An optional switch that applies to `active_high_latch`, `active_low_latch`, `posedge_dff`, `negedge_dff`, `posedge_scan_cell`, `negedge_scan_cell`, `posedge_synchronizer_cell` or `negedge_synchronizer_cell` and specifies to return a cell with an asynchronous input to reset the cell’s state to “0” (low). For all types except `posedge_synchronizer_cell` and `negedge_synchronizer_cell`, if a cell with exactly what is requested does not exist, null will be returned. However, for the synchronizer cells, a cell will be returned if it has all the

features requested, and can be tied off outside to produce the requested cell. For example, if a synchronizer_cell with no asynchs is requested (no -with_set or -with_reset switches) and none exists, but a sync cell with one or more asynchs does exist, it will be returned.

Similarly, if a sync_cell -with_reset is requested, and none with only a reset exists, but one with both asynchs does exist, it will be returned.

- **-length *length***

A switch and integer pair that specifies the length of a posedge_synchronizer_cell or a negedge_synchronizer_cell. The length is the number of DFFs in the synchronizer cell and can be specified using a single number or a number:number range. If the dft_cell_selection does not contain a synchronizer_cell of the length specified, then an empty string will be returned.

- **-cell_selection_name *dft_cell_selection_name***

A switch and string pair that specifies the name of a currently available cell selection to retrieve cells from. This could be a technology name, for example. This switch is optional if only one dft_cell_selection wrapper is currently loaded. If this switch is not specified and more than one dft_cell_selection wrapper exists, the tool issues an error. Similarly, if an invalid technology is specified, the tool issues an error and lists all available dft_cell_selection names.

- **-inputs *number***

An optional switch and integer pair that specifies the number of inputs on the model. The default number is 2. The default is ignored except in the case of symmetrical primitives {and, nand, or, nor, xor}. The default for symmetrical primitives is 2 unless the -inputs number pair specifies an integer greater than 2; in this case the tool returns a cell with the specified number of inputs if one exists in the specified technology.

- **-with_inverted_inputs *number_inverted_inputs***

An optional switch and integer pair that specifies the number of inverted inputs of the total inputs specified with the -inputs switch. At least one input must be non-inverting. This switch can only be used when *function_name* is “and” or “or”.

- **-silent**

An optional switch that specifies to suppress the error message and returns an empty collection.

Return Values

Returns the module objects when no errors are present.

Examples

Example 1

In the following example, the command queries which cell model is available in the tech_name_1 dft_cell_selection wrapper to be used as a clock_buffer.

```
get_dft_cell clock_buffer -cell_selection_name tech_name_1
```

```
{my_buff}
```

Example 2

In the following example, the command queries which 4-input cell model is available to be used as an and gate.

```
get_dft_cell and -input 4
{and_4}
```

Example 3

In the following example, multiple dft_cell_selection wrappers exist but the -cell_selection_name switch is not specified. As a result, the tool returns an error and lists the available dft_cell_selection names that exist in the library.

```
get_dft_cell and
// Error: Multiple cell_selection(<selection_name>) exist.
// Must specify -cell_selection_name followed by one of :
//   tech_name_1
//   tech_name_2
//   tech_name_3
```

Example 4

In the following example, an invalid technology is specified. As a result, the tool returns an error and lists the available technologies.

```
get_dft_cell and -cell_selection_name foo
// Error: cell_selection(foo) does not exist.
// -cell_selection_name must currently be followed by one of :
//   tech_name_1
//   tech_name_2
//   tech_name_3
```

Example 5

In the following example, a procedure called intercept_with_cell_body is created to allow intercepting ports, nets, or pins that are either sinks or drivers. The second argument is a cell_selection type string that uses the get_dft_cell command to obtain the corresponding cell from the library, thereby making this command technology-independent. To understand the purpose of each section, read the comments in the commented section headers.

Note, in Section 3, the port attribute called ‘function’ is used to determine the port names to connect on the selected dft cell. You can also refer to the example for the [register_tcl_command](#) to see how you can create a cleaner external interface to this command with proper option parsing and help messages.

```
#####
### This array is to not repeat interception into repeated module
### instances when auto_uniquify_edited_modules is off
#####

variable insertion_parent_leaf_instance_array
array set insertion_parent_leaf_instance_array {}
proc intercept_with_cell_body {args} {

    # Initialize the allArgs array with the default values of the optional
    # options
    array set allArgs {
        input2 ""
        select ""
        level 1
        clear 0
    }

    #Override the default values of the arguments with the actual values
    # received from
    array set allArgs $args
    set node $allArgs(node)
    set cell_sel $allArgs(cell_sel)
    set in2 $allArgs(input2)
    set sel $allArgs(select)
    set level $allArgs(level)
    set clear $allArgs(clear)

#####

##### Section 1: Defines a global array to know if an interception was
##### already done inside repeated module instance. clear args set to 1
##### resets the array
#####

variable insertion_parent_leaf_instance_array

if {$clear == 1} {
    array unset insertion_parent_leaf_instance_array
    array set insertion_parent_leaf_instance_array {}
}
set dft_cell [get_dft_cell $cell_sel]
##### Section 2: Validates node argument and determine if it is a sink or
##### driver interception
#####

set node_object [check_obj_spec $node <node>]
set node_type [get_attribute_value_list $node_object \
    -name object_type]
if {$node_type eq "port" } {
    if {[get_attribute_value_list $node_object -name direction] \
        eq "input"} {
        set insertion_type "driver"
    } else {
        set insertion_type "sink"
    }
    set leaf_name [lindex [get_attribute_value $node_object \
        -name name] 0]
```

```

        set instance_object ""
        set leaf_block_id ""
    } elseif {$node_type eq "pin"} {
        if {[get_attribute_value_list $node_object -name direction] \
            eq "output"} {
            set insertion_type "driver"
        } else {
            set insertion_type "sink"
        }
        set instance_object [get_instances -of_pins $node_object]
        set leaf_name [lindex [get_attribute_value $node_object \
            -name leaf_name] 0]
        set leaf_block_id [lindex [get_attribute_value $instance_object \
            -name leaf_block_id] 0]
    } else {
        set insertion_type "driver"
        set instance_object [get_instances -of_nets $node_object]
        set leaf_name [lindex [get_attribute_value $node_object \
            -name leaf_name] 0]
        set leaf_block_id [lindex [get_attribute_value $node_object \
            -name leaf_block_id] 0]
    }
}

#####
### Section 3: Determine if in2 and sel are needed
#####

set connect_in2 0
set connect_sel 0
if {$cell_sel == "mux" | $cell_sel == "clock_mux"} {
    set input0 "[get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == mux_in0}]]"
    set input1 "[get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == mux_in1}]]"
    set select "[get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == mux_select}]]"
    set output "[get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == data_out}]]"
    set connect_in2 1
    set connect_sel 1
} elseif {$cell_sel == "buffer" | $cell_sel == "clock_buffer" | \
    $cell_sel == "inverter"} {
    set input0 "[ get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == data_in}]]"
    set output "[ get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == data_out}]]"
} else {
    set input0 "[lindex [get_name_list [get_ports -of_modules $dft_cell \
        -filter {function == data_in}]] 0]"
    set input1 "[lindex [get_name_list [get_ports -of_modules $dft_cell \
        -filter {function == data_in}]] 1]"
    set output "[get_single_name [get_ports -of_modules $dft_cell \
        -filter {function == data_out}]]"
    set connect_in2 1
}
if {$connect_in2} {
    if {$in2 != ""} {

```

```

        set in2_object [check_obj_spec $in2 "-input2"]
    } else {
        return -code error "-input2 must be specified when \
            -cell_function_name is two input cell"
    }
}

if {$connect_sel} {
    if {$sel != ""} {
        set sel_object [check_obj_spec $sel "-select"]
    } else {
        return -code error "-select must be specified when \
            -cell_function_name is mux or clock_mux"
    }
}

#####
### Section 4: Determines location of the interception gate and if it
### was already done for the specified node inside its parent module.
#####

if {[sizeof_collection $instance_object] ne 0} {
    if {$node_type eq "net"} {
        set leaf_instance_name ""
        set parent_module [get_modules -of_instances $instance_object]
        set parent_instance_name [get_single_name $instance_object]
    } else { # it is a pin
        set leaf_instance_name [lindex [get_attribute_value_list \
            $instance_object -name leaf_name_hash] 0]
        set parent_instance [get_instances -parent_of_instances \
            $instance_object]
        set parent_instance_name [get_single_name $parent_instance]
        if {[sizeof_collection $parent_instance] eq 0} {
            set parent_module [get_current_design]
        } else {
            set parent_module [get_modules -of_instances \
                $parent_instance]
        }
    }
} else {
    set parent_instance_name ""
    set leaf_instance_name ""
    set parent_module [get_current_design]
}
set parent_mod_name [get_single_name $parent_module]
set parent_lib_name [lindex [get_attribute_value_list $parent_module \
    -name library_name] 0]
set parent_arch_name [lindex [get_attribute_value_list $parent_module \
    -name vhdl_architecture] 0]

#####
# This key will match for insertion made inside repeated instances of
# a module or inside a generate loop block. When it match, the
# interception is skipped for the next invocation
#####

set key "Level${level}-Cell(${cell_sel})-"
append key \

```

```

"${parent_lib_name}::${parent_mod_name}(${parent_arch_name})"
if {$leaf_instance_name ne ""} {
    append key "/${leaf_instance_name}"
}
append key "/${leaf_name}"

#####
#### Section 5: Retrieves leaf instance name of gate if node is a
#### repeated module instance and skips the interception
#####

if {[info exists insertion_parent_leaf_instance_array($key)] && \
     [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
    set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
    if {$leaf_block_id ne ""} {
        set cell_inst [get_instance \
                       "${parent_instance_name}/$leaf_block_id}.${leaf_cell_inst}"]
    } else {
        set cell_inst [get_instance \
                       "${parent_instance_name}/$leaf_cell_inst"]
    }
} else {
    if {$leaf_block_id ne ""} {
        set cell_inst_name \
        "${parent_instance_name}/$leaf_block_id.insertion_${cell_sel}"
    } else {
        set cell_inst_name \
        "${parent_instance_name}/insertion_${cell_sel}"
    }
    no_transcript {
        set cell_inst [create_instance $cell_inst_name -of_modules \
                      $dft_cell -allow_instance_uniquification]
    }
}

set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
                      leaf_name] 0]
set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
                           -name leaf_block_id] 0]
if {$leaf_block_id ne ""} {
    # Remove the block_id part from the leaf instance name
    set leaf_name [string range $leaf_name \
                  [string length $leaf_block_id] end]
}
set insertion_parent_leaf_instance_array($key) $leaf_name

if {$insertion_type == "sink"} {
    no_transcript {
        move_connections -from $node_object \
                          -to [get_pins ${input0} -of_instances $cell_inst]
        create_connections [get_pins ${output} -of_instances \
                           $cell_inst] $node_object
    }
} else {
    no_transcript {
        move_connections -from $node_object -to [get_pins ${output} \
                                                -of_instances $cell_inst]
        create_connections $node_object [get_pins ${input0} \
                                         -of_instances $cell_inst]
    }
}

```

```

        }
    }

    if {$connect_in2} {
        no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
                $cell_inst]
        }
    }

    if {$connect_sel} {
        no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
                $cell_inst]
        }
    }

    return $cell_inst
}

#####
# This procs checks that the value is either an single element collection
# of nets, ports, or pins or the name of a single net, port, or pin.
#####

proc check_obj_spec {obj_or_name id} {
    if {[is_collection $obj_or_name]} {
        if {[sizeof_collection $obj_or_name] ne 1} {
            display_message "Supplied collection for ${id} has \
                more than one element."
            return -code error
        } else {
            set obj_type [lindex [get_attribute_value_list $obj_or_name \
                -name object_type] 0]
            set object $obj_or_name
            if {$obj_type ne "port" && $obj_type ne "pin" && \
                $obj_type ne "net"} {
                display_message "Specified value for ${id} has \
                    type '$obj_type' instead of 'pin', 'port' or 'net'!"
                return -code error
            }
        }
    } else {
        set net_object [get_nets $obj_or_name -silent]
        if {[sizeof_collection $net_object] eq 0} {
            display_message "Specified value for ${id} is not an \
                existing 'pin', 'port' or 'net' name."
            return -code error
        } elseif {[sizeof_collection $net_object] ne 1} {
            display_message "Specified value for ${id} is more \
                than one bit wide"
            return -code error
        }
        set port_object [get_ports $obj_or_name -silent]
        set pin_object [get_pins $obj_or_name -silent]
        if {[sizeof_collection $port_object] ne 0} {
            #If the name is also a port, pick the port object
            set object $port_object
        }
    }
}

```

```
    } elseif {[sizeof_collection $pin_object] ne 0} {
        #If the name is also a pin, pick the pin object
        set object $pin_object
    } else {
        #The net is not a port nor a pin
        set object $net_object
    }
}
return $object
}
```

Related Topics

[report_context](#)

[set_context](#)

[get_dft_info_dictionary](#)

Context: dft, patterns

Mode: setup, analysis

Creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion.

Usage

```
get_dft_info_dictionary [-example_usage_script]
```

Description

This command creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion. When you are using Tesson Scan for scan insertion, the information about the DFT logic you inserted into the RTL is automatically understood and used during scan insertion. If you are performing the scan insertion as part of synthesis, the dictionary created by this command can be used to import the information into the synthesis tool.

Note that the DFT info dictionary is automatically stored into the [dft_inserted_designs](#) directory when the [extract_icl](#) command is called with the `-write_in_tsdb` switch set or defaulting to on.

The format of the dictionary is illustrated in the syntax summary below. The version, dft_signals, modules_with_chains, non_scannable_instance_list and edt_instances keys always exist even if their values are empty:

```

set tessent_dft_info_dict {
    version 1
    dft_signals {
        dft_signal_name {
            connection_node_name node_name
            connection_node_type port|pin
            forced_value_in_pre_scan_drc 0|1
        }
    }
    modules_with_chains {
        module_name {
            pre_scan_drc_on_boundary_only 0|1
            module_type normal|memory|occ
            is_hard_module 0|1
            internal_scan_only 0|1
            allow_scan_out_retiming 0|1
            instance_list {inst_name ...}
            scan_en_ports|ltest_en_ports|set_reset_ports {
                port_name {
                    active_polarity 0|1
                }
                clock_ports {
                    port_name {
                        off_state 0|1
                    }
                }
                clock_out_ports_or_pins {
                    port_or_pin_name {}
                }
            }
            scan_chains {
                scan_chain_name {
                    length auto|int
                    scan_in_port port_name
                    scan_in_clock_name port_name
                    scan_in_clock_inversion 0|1
                    scan_out_port port_name
                    scan_out_clock_name port_name
                    scan_out_clock_inversion 0|1
                }
            }
        }
    }
    non_scannable_instance_list {inst_name ...}
    edt_instances {
        instance_name {
            edt_module_name module_name
            scan_chains {
                chain_name {
                    scan_in_port port_name
                    scan_out_port port_name
                }
            }
        }
    }
}

```

```

        }
    }
}
```

Arguments

- **-example_usage_script**

An optional Boolean switch used to request an example usage script that you can use as a starting example to convert the dictionary into the specific commands of your synthesis tool.

The places you are expected to enter tool-specific commands are identified with the **###** comments. The content of the example usage script is shown in the Example below.

Examples

The following example runs the `get_dft_info_dictionary` command to create an example usage script to be used with the DFT info dictionary created as part of the `extract_icl` command:

```

process_dft_specification
extract_icl

> process_dft_specification
> extract_icl// Begin ICL extraction.
// -----
// ICL extraction completed, ICL instances=13, CPU time=0.20 sec.
// -----
// -----
// Begin ICL elaboration and checking.
// -----
// ICL elaboration completed, CPU time=0.12 sec.
// -----
// Writing ICL file : ./tsdb_outdir/dft_inserted_designs/
corea_rtl2.dft_inserted_design/corea.icl
// Writing and sourcing PDL file : ./tsdb_outdir/dft_inserted_designs/
corea_rtl2.dft_inserted_design/corea.pdl
// Writing SDC file: ./tsdb_outdir/dft_inserted_designs/
corea_rtl2.dft_inserted_design/corea.sdc
// Writing DFT info dictionary : ./tsdb_outdir/dft_inserted_designs/
corea_rtl2.dft_inserted_design/corea.dft_info_dictionary
```

```

> puts [get_dft_info_dictionary -example_usage_script]
puts "Processing DFT signals"
set dft_signals_dict [dict get $tessent_dft_info_dict dft_signals]
puts "Setting up Static Dft Signals"
foreach dft_signal [dict keys $dft_signals_dict] {
    if {[dict exists $dft_signals_dict $dft_signal] \
        forced_value_in_pre_scan_drc} {
        set connection_node_name \
            [dict get $dft_signals_dict $dft_signal connection_node_name]
        set connection_node_type \
            [dict get $dft_signals_dict $dft_signal connection_node_type]
        set forced_value_in_pre_scan_drc \
            [dict get $dft_signals_dict $dft_signal forced_value_in_pre_scan_drc]
        puts "--- Static Dft Signal $dft_signal ---"
        if {$connection_node_type eq "pin"} {
            ### Command to cut and force created pseudo port
            ###
        } else {
            ### Command tp force port
            ###
        }
    }
}
puts "Setting Dynamic Dft Signals"

array set scan_signal_map {ltest_en -ten shift_capture_clock -tclk}
foreach dft_signal {scan_en ltest_en shift_capture_clock} {
    if {[dict exists $dft_signals_dict $dft_signal]} {
        set connection_node_name \
            [dict get $dft_signals_dict $dft_signal connection_node_name]
        set connection_node_type \
            [dict get $dft_signals_dict $dft_signal connection_node_type]
        ### Command to declare signal as the source of the DFT function goes here
        ###
    }
}

```

```

puts "Declare the Non-Scannable instances"

set non_scannable_instance_list \
    [dict get $tessent_dft_info_dict non_scannable_instance_list]
if {[llength $non_scannable_instance_list] > 0} {
    ### Command to declare the Non-Scannable instances
    ###
}puts "EDT Scan chain connections"

set edt_instances_dict [dict get $tessent_dft_info_dict edt_instances]
foreach edt_instance [dict keys $edt_instances_dict] {
    set scan_chains_dict [dict get $edt_instances_dict $edt_instance scan_chains]
    foreach scan_chain [dict keys $scan_chains_dict] {
        set scan_in_port [dict get $scan_chains_dict $scan_chain scan_in_port]
        set scan_out_port [dict get $scan_chains_dict $scan_chain scan_out_port]
        ### Command to declare scan chains SI/SO locations
        ###
    }
}
puts "Declare sub-chains and extract OCC clocks"

puts "Declare sub-chains and extract OCC clocks"
set occ_clock_pin_list [list]
set modules_with_chains_dict \
    [dict get $tessent_dft_info_dict modules_with_chains]
foreach module_with_chain [dict keys $modules_with_chains_dict] {
    set instance_list \
        [dict get $modules_with_chains_dict $module_with_chain instance_list]
    set module_type \
        [dict get $modules_with_chains_dict $module_with_chain module_type]
    if {$module_type eq "occ"} {
        set clock_out_ports_or_pins_dict \
            [dict get $modules_with_chains_dict $module_with_chain clock_out_ports_or_pins]
        foreach clock_out [dict keys $clock_out_ports_or_pins_dict] {
            foreach instance $instance_list {
                lappend occ_clock_list "${instance}/${clock_out}"
            }
        }
    }
}

```

```

set scan_en_port_list [list]
set scan_en_ports_dict \
[dict get $modules_with_chains_dict $module_with_chain scan_en_ports]
foreach scan_en_port [dict keys $scan_en_ports_dict] {
    set active_polarity \
[dict get $scan_en_ports_dict $scan_en_port active_polarity]
    ### Command to declare the Scan enable ports
    ###
}   set ltest_en_port_list [list]
if {[dict exists $modules_with_chains_dict $module_with_chain ltest_en_ports]} {
    set ltest_en_ports_dict \
[dict get $modules_with_chains_dict $module_with_chain ltest_en_ports]
    foreach ltest_en_port [dict keys $ltest_en_ports_dict] {
        set active_polarity \
[dict get $ltest_en_ports_dict $ltest_en_port active_polarity]
        ### Command to declare the ltest enable ports
        ###
    }
}

set set_reset_port_list [list]
if {[dict exists $modules_with_chains_dict $module_with_chain set_reset_ports]}{
    set set_reset_ports_dict \
[dict get $modules_with_chains_dict $module_with_chain set_reset_ports]
    foreach set_reset_port [dict keys $set_reset_ports_dict] {
        set active_polarity \
[dict get $set_reset_ports_dict $set_reset_port active_polarity]
        ### Command to declare the ltest enable ports
        ###
    }
}

set clock_port_list [list]
if {[dict exists $modules_with_chains_dict $module_with_chain clock_ports]} {
    set clock_ports_dict \
[dict get $modules_with_chains_dict $module_with_chain clock_ports]
    foreach clock_port [dict keys $clock_ports_dict] {
        set off_state [dict get $clock_ports_dict $clock_port off_state]
        ### Command to declare the input clock ports
        ###
    }
}

```

```

set chain_cnt 0
set scan_chains_dict \
    [dict get $modules_with_chains_dict $module_with_chain scan_chains]
foreach scan_chain [dict keys $scan_chains_dict] {
    if {[![dict exists $scan_chains_dict $scan_chain scan_in_clock_name] || \
        ![dict exists $scan_chains_dict $scan_chain scan_out_clock_name]]} {
        puts "// Warning: Ignoring Chain '$scan_chain' in Module"
        puts "// '$module_with_chain' because its scan_in_clock or scan_out_clock"
        puts "// ports are not define and this tool is not able to extract it."
        continue
    }
    set length [dict get $scan_chains_dict $scan_chain length]
    if {$length eq "auto"} {
        puts "Warning: Using length of 8 for Chain $scan_chain of Module"
        puts "$module_with_chain because it had a value of auto"
        puts "          And this tool does nto support sub chain tracing to extract"
        puts "it. It may affect chain balancing."
        set length 8
    }
    set scan_in_port  [dict get $scan_chains_dict $scan_chain scan_in_port]
    set scan_in_clock_name \
        [dict get $scan_chains_dict $scan_chain scan_in_clock_name]
    set scan_in_clock_inversion \
        [dict get $scan_chains_dict $scan_chain scan_in_clock_inversion]
    set scan_out_port \
        [dict get $scan_chains_dict $scan_chain scan_out_port]
    set scan_out_clock_name \
        [dict get $scan_chains_dict $scan_chain scan_out_clock_name]
    set scan_out_clock_inversion \
        [dict get $scan_chains_dict $scan_chain scan_out_clock_inversion]

    ### Command to declare the sub chain
    ###

    incr chain_cnt
}
}
puts "Declaring OCC output clocks"
foreach occ_clock $occ_clock_list {
    ### Command to declare clocks with off state of 0
    ###
}

```

get_dft_signal

Context: dft, patterns

Mode: setup, analysis, insertion

Obtains the source of a DFT signal or various aspects about the DFT signal.

Usage

```
get_dft_signal {-list | -source_node pin_port_net_spec
               | name [-unmapped_across_pad | -exists | -list_of_parent_instances
               | -scan_mode_type] [-parent_instance parent_instance] }
               [ -usage usage ] [ -silent ]
```

Description

Obtains the source of a DFT signal or various aspect about the DFT signal. Between the time a DFT signal was added and process_dft_specification has completed its run, the existence of a DFT signal can be checked with the -exists switch, but the signal's location cannot be obtained until it is actually created.

You can query the location of a DFT signal within the [process_dft_specification.post_insertion](#) callback as the DFT signal is created before the proc is called. As you can see in the example below, the get_dft_signal command is useful at several stages of the flow to obtain the location of a previously added DFT signal.

Arguments

- **-list**

A Boolean switch that specifies to return the list of existing or to-be-created DFT signals. The switch can be combined with the -usage switch to limit the list to a single usage as shown in the example.

When the -list, -exists, -list_of_parent_instances, and -scan_mode_type switches are not specified, the command returns a collection containing the object on which the DFT signal exists.

- **-source_node *pin_port_net_spec***

A switch-value pair used to specify the name of a single port, pin, net object or a collection containing a single port, pin, net object. When used, the command returns the name of the DFT signal that was added on the specified node. If no DFT signal were added on the specified node, an error is generated unless the -silent switch is used in which case the error message is suppressed and an empty string is returned.

- ***name***

A string argument used to specify the name of a DFT signal for which you want its location or other aspect such as its existence, or the type of scan_mode it is.

- **-unmapped_across_pad**

A Boolean switch used when a DFT signal is specified on a port connected to a pad cell. In this situation, the signal is mapped to the other side of the pad cell. When this switch is specified, the unmapped location is returned. If the signal is not connected to a pad cell and this switch is specified, it has no effect and the unmapped location is reported as usual.

- **-exists**

A Boolean switch used to specify that a 1 should be returned if the specified DFT signal name exist and when the -usage switch is also specified that the DFT signal is of the specified usage. A “0” is returned otherwise.

- **-list_of_parent_instances**

A Boolean switch that requests the list of parent instances in which the DFT signal exists. A null value in the list represents the top module. This option is only valid when combined with the name edt_clock and shift_capture_clock as those are the only two signals that may have several copies located in different parent instances. See the [add_dft_signals](#) -parent_instance switch description for more details.

- **-scan_mode_type**

A Boolean switch used to request the type of DFT signal having the usage scan_mode. The returned values are unwrapped, internal, external, or retargeting when the DFT signal has the usage scan_mode. The return value is a null string when the DFT signal has a usage other than scan_mode.

- **-parent_instance *parent_instance***

A switch-value pair used to specify a parent instance in which the DFT signal is to be located. This option is only valid when combined with the name edt_clock and shift_capture_clock as those are the only two signals that may have several copies located in different parent instances. See the [add_dft_signals](#) -parent_instance switch description for more details.

- **-usage *usage***

A switch-value pair that is used to limit the DFT signals to a specified usage. The five allowed usages are: global_dft_control, logic_test_control, scan_mode, dynamic, static_ijtag. The usage static_ijtag means the DFT signal was added with -type static_dft_control and with either the “-create_with_tdr” switch or with the “-source_node *port* -make_ijtag_port” switches which makes it controllable by ijtag.

When combined with the -list switch, it restricts the list to the DFT signal of the specified usage. When combined with the -exists switch, the command only returns “1” if the DFT signal exists and is of the specified usage.

- **-silent**

A Boolean switch that suppresses the error message that is normally generated when the specified DFT signal does not exist or is not of the requested usage. Instead of generating an error, an empty collection is silently returned.

Examples

The following example illustrates the usage of the various switches:

```

set_context dft -rtl
read_verilog ..../data/corea.v
set_current_design corea
set_design_level physical_block
add_dft_signals ltest_en int_ltest_en ext_ltest_en \
    memory_bypass_en async_set_reset_static_disable \
    int_mode ext_mode

add_dft_signals scan_en test_clock edt_update \
    -source_nodes {my_scan_en my_test_clock my_edt_update}

add_dft_signals edt_clock shift_capture_clock \
    async_set_reset_dynamic_disable \
    -create_from_other_signals

check_design_rules
create_dft_specification
process_dft_specification

foreach usage {global_dft_control logic_test_control \
    scan_mode dynamic} {
    foreach dft_signal [get_dft_signal -list -usage $usage] {
        puts "get_dft_signal $dft_signal -exists ="
        puts "[get_dft_signal $dft_signal -exists]"
        puts "get_dft_signal $dft_signal ="
        puts "[get_single_name [get_dft_signal $dft_signal]]"
        if {$usage eq "scan_mode"} {
            puts "get_dft_signal $dft_signal -scan_mode_type ="
            puts "[get_dft_signal $dft_signal -scan_mode_type]"
        }
    }
}
get_dft_signal async_set_reset_static_disable -exists =
1

get_dft_signal async_set_reset_static_disable =
corea_rtl_tessent_tdr_sri_ctrl_inst/async_set_reset_static_disable

get_dft_signal ext_ltest_en -exists =
1

get_dft_signal ext_ltest_en =
corea_rtl_tessent_tdr_sri_ctrl_inst/ext_ltest_en

```

```
get_dft_signal int_ltest_en -exists =
1

get_dft_signal int_ltest_en =
corea_rtl_tessent_tdr_sri_ctrl_inst/int_ltest_en

get_dft_signal ltest_en -exists =
1

get_dft_signal ltest_en =
corea_rtl_tessent_tdr_sri_ctrl_inst/ltest_en

get_dft_signal memory_bypass_en -exists =
1

get_dft_signal memory_bypass_en =
corea_rtl_tessent_tdr_sri_ctrl_inst/memory_bypass_en

get_dft_signal ext_mode -exists =
1

get_dft_signal ext_mode =
corea_rtl_tessent_tdr_sri_ctrl_inst/ext_mode

get_dft_signal ext_mode -scan_mode_type =
external

get_dft_signal int_mode -exists =
1

get_dft_signal int_mode =
corea_rtl_tessent_tdr_sri_ctrl_inst/int_mode

get_dft_signal int_mode -scan_mode_type =
internal

get_dft_signal async_set_reset_dynamic_disable -exists =
1

get_dft_signal async_set_reset_dynamic_disable =
tessent_persistent_cell_async_set_reset_dynamic_disable/y

get_dft_signal edt_clock -exists =
1
```

```

get_dft_signal edt_clock =
tessent_persistent_cell_edt_clock/clkg

get_dft_signal edt_update -exists =
1

get_dft_signal edt_update =
my_edt_update

get_dft_signal input_wrapper_scan_en -exists =
1

get_dft_signal input_wrapper_scan_en =
tessent_persistent_cell_input_wrapper_scan_en/y

get_dft_signal output_wrapper_scan_en -exists =
1

get_dft_signal output_wrapper_scan_en =
tessent_persistent_cell_output_wrapper_scan_en/y

get_dft_signal scan_en -exists =
1

get_dft_signal scan_en =
my_scan_en

get_dft_signal shift_capture_clock -exists =
1

get_dft_signal shift_capture_clock =
tessent_persistent_cell_shift_capture_clock/clkg

get_dft_signal test_clock -exists =
1

get_dft_signal test_clock =
my_test_clock

```

Error message when the requested signal does not exist:

```

catch {get_dft_signal junk}
// Error: The DFT signal named 'junk' does not exist. Run the
// 'report_dft_signals' command for more information.

```

Error message when the requested signal is of the wrong usage:

```
catch {get_dft_signal ltest_en -usage scan_mode}  
// Error: The DFT signal named 'ltest_en' is for usage  
// 'logic_test_control' instead of 'scan_mode'.
```

Error message suppressed when -silent is used:

```
get_dft_signal ltest_en -usage scan_mode -silent
```

The -unmapped_across_pad parameter usage and comparison:

```
add_dft_signals scan_en -source_node gpio3  
get_dft_signals scan_en  
{gpio3_pad/o}  
get_dft_signal scan_en -unmapped_across_pad  
{gpio3}
```

Related Topics

[add_dft_signals](#)
[process_dft_specification](#)
[report_dft_signals](#)
[set_static_dft_signal_values](#)

get_dft_specification_requirement

Context: dft patterns

Mode: all modes

Returns the requirements set using the [set_dft_specification_requirements](#) command.

Usage

```
get_dft_specification_requirement -memory_bist | -memory_bisr_chains |  
-memory_bisr_controller | -boundary_scan | -ac_boundary_scan | -logic_test
```

Description

Returns the requirements set using the [set_dft_specification_requirements](#) command.

Arguments

- **-memory_bist**
An option that requests the value of the [set_dft_specification_requirements](#) -memory_bist option.
- **-memory_bisr_chains**
An option that requests the value of the [set_dft_specification_requirements](#) -memory_bisr_chains option.
- **-memory_bisr_controller**
An option that requests the value of the [set_dft_specification_requirements](#) -memory_bisr_controller option.
- **-boundary_scan**
An option that requests the value of the [set_dft_specification_requirements](#) -boundary_scan option.
- **-ac_boundary_scan**
An option that requests the value of the [set_dft_specification_requirements](#) -ac_boundary_scan option.
- **-logic_test**
An option that requests the value of the [set_dft_specification_requirements](#) -logic_test option.

Examples

The following example queries the value of the -memory_test option.

```
get_dft_specification_requirements -memory_bist  
off  
set_dft_specification_requirements -memory_test on
```

get_dft_specification_requirements -memory_bist

auto

Related Topics

[set_dft_specification_requirements](#)

get_drc_handling

Context: unspecified, all contexts

Mode: all modes

Retrieves the DRC handling status for the specified *drc_name*.

Usage

```
get_drc_handling drc_name [-auto_fix]
```

Description

Retrieves the DRC handling status for the specified *drc_name*.

Arguments

- *drc_name*

A required string that specifies the name of the of the DRC you for which you want the DRC handling status.

- -auto_fix

An optional switch that enables you to see if the DRC supports auto fixing of the violation. If you registered your own custom DRC using the [register_drc](#) command and you used the -allow_auto_fix switch, this switch will return 1 unless you explicitly turned off auto fixing using the “[set_drc_handling](#) -auto_fix” command. It will always return 0 for a DRC which does not allow auto fixing. You use this command in the body of your custom DRC to know if the violation should be reported or auto fixed.

[get_equivalent_editable_node](#)

Context: dft

Mode: setup, analysis, insertion

Checks if a node is editable. If the node is not editable, then returns the node's editable equivalent node if it exists.

Usage

`get_equivalent_editable_node node_spec`

Description

The `get_equivalent_editable_node` command accepts a pin, port, net, pseudo_port or gate_pin objects as its argument.

If the object is a pseudo_port or a gate_pin, the tool maps it to its corresponding port, pin, or net object. If the object is a pin which is on or inside an instance created by RTLC, or inside a hard module or a cell or if the object is a net which is inside an instance created by RTLC, or inside a hard module or a cell, the tool maps it to an equivalent node in the editable region if one exists.

If the object cannot be mapped to an editable object, an empty collection is returned. Use this command when you are using the synthesized view to perform some analysis and then you want to perform edits on the RTL view.

Arguments

- ***node_spec***

A required string used to specify the name of a pin, port, net, pseudo_port or gate_pin object or a collection containing a single pin, port, net, pseudo_port or gate_pin object.

Examples

The following example shows the use of the command to map various nodes to its editable equivalent:

Pin on a synthesized instance:

```
set map [get_equivalent_editable_node ex2_cell/rtlc0_1/sel_0]
set type [get_attribute_value_list $map -name object_type]
puts "$type [get_single_name $map]"
pin ex2_cell/c
```

Pin on a normal instance:

```
set map [get_equivalent_editable_node ex2_cell/a]  
set type [get_attribute_value_list $map -name object_type]  
puts "$type [get_single_name $map]"  
pin ex2_cell/a
```

Pseudo_ports are mapped to their associated design objects:

```
puts [get_single_name [get_equivalent_editable_node pseudo_a]]  
ex1_i1/a
```

A gate_pin is mapped to its pseudo port then to the pin:

```
puts [get_single_name [get_equivalent_editable_node  
ex1_i1/y
```

get_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fanin of the specified pin, net, or port object.

Usage

```
get_fanins obj_spec [-stop_on {cell_pin | net | pin}] [-silent]
```

Description

Returns a collection of all requested objects found in the fanin of the specified pin, net, or port object.

When *obj_spec* is an output or inout pin of a cell, then the command reports all input or inout pins on the same cell that are in the combinational fanin of the output or inout pin. Note that the fanin direction of an inout pin is toward the inside of the instance, and the fanout direction of an inout pin is toward the outside of the instance.

The -stop_on option specifies object types upon which the tracing should stop. It does not mean that the returned collections contain only objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net will be reported as the object on the fanin of the pin even if -stop_on is specified as cell_pin. If the net had not been dangling and it had a driver, the get_fanins command would not have stopped on the net and would have continued to walk the fanin path.

The get_fanins command returns an empty collection when there is no fanin. The fanin of an input port is null. When the fanin of an input pin is an empty collection, the pin is either unconnected or tied. Use the tie_value attribute to see it if is tied to 0 or 1 or unconnected.

The get_fanins command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the [trace_flat_model](#) command to perform tracing on the flat model with simulation conditions considered.

Arguments

- *obj_spec*

A required value that specifies a Tcl list of one or more net, pin, port, or pseudo_port names; or a collection of one or more net, pin, port, or pseudo_port objects.

- -stop_on {cell_pin | net | pin}

An optional switch and literal pair that specifies the following:

- When -stop_on is set to cell_pin (the default), the fanin walk traverses nets and design pins but stops on pins of cells or primitives in the fanin of the object. The

walk may still, however, stop on a net or a design pin when the net or design pin does not have a fanin.

- When -stop_on is set to net, the fanin walk traverses through one level of “assign” statements and stops on the net connected to the start pin.
- When -stop_on is set to pin, the fanin walk traverses nets and reports the pin in the fanin of the object. This could be an input or inout pin above the object or an output or inout pin on child instances. The walk may still, however, stop on a net if the net does not fanin to pins.
- -silent

An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Examples

Example 1

The following example shows a series of get_fanins command invocations of the circuit example shown in [Figure 4-3](#).

```
# find net connected to pin u4/I5
get_fanins u4/I5 -stop_on net
{n3}

#Find cell_pin in the fanin of u4/I5
get_fanins u4/I5
{u3/Y}

#Find pin in the fanin of u4/I5. Returns cell pins as there is no design pins before it
get_fanins u4/I5 -stop_on pin
{u3/Y}

#Find pin in the fanin of u4/u1/A0
get_fanins u4/u1/A0 -stop_on pin
{u4/I1}

#Find all nets connected to the D pin of dff modules
get_fanins [get_pins -filter {module_name == "dff" && leaf_name == "D"}] -stop_on net
{I3 n6 u4/n2 u4/n3 u4/u5/n1 u4/u5/I3 u5/n1}

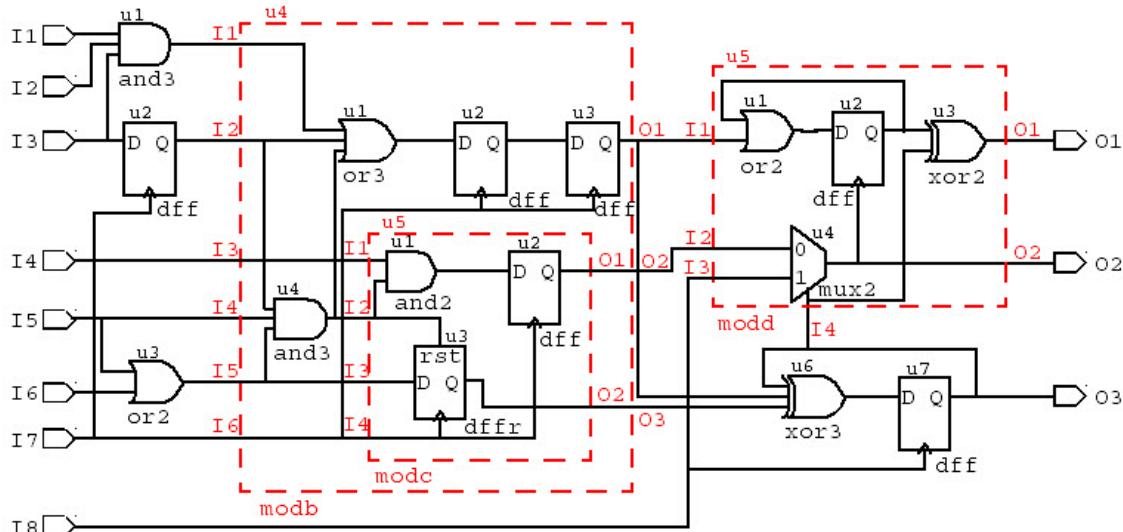
#Find pins in the fanin of an output pin of a cell
get_fanins u1/Y
{u1/A0 u1/A1 u1/A2}
```

Example 2

The following example adds an internal primary input (pseudo_port). Based on the example circuit shown in [Figure 4-3](#), it then returns the collection of objects found in the fanin of both of the pseudo_port's underlying pins.

```
add_primary_inputs {u4/I4 u4/I5} -internal -pseudo_port_name user_pi1
get_fanins user_pi1
{I5 u3/Y}
```

Figure 4-3. Circuit Illustrating the Usage of the get_fanins Command



Related Topics

[get_icl_ports](#)
[get_nets](#)

get_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.

Usage

```
get_fanouts obj_spec [-stop_on cell_pin | net | pin] [-silent]
```

Description

Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.

When *obj_spec* is an input pin of a cell, then the command reports all output or inout pins on the same cell that are in the combinational fanout of the input pin. Note that the fanout direction of an inout pin is toward the outside of the instance, and the fanin direction of an inout pin is toward the inside of the instance.

The -stop_on option is used to specify object types upon which the tracing should stop. Specifying this option does not mean that the returned collections will only contain objects of the specified type, because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net will be reported as the object on the fanout of the pin even if -stop_on is specified as cell_pin. If the net had been connected to other pins, the get_fanouts command would not have stopped on the net and would have continued to walk the fanout path.

The get_fanouts command returns an empty collection when there is no fanout. The fanout of an output or inout port is null.

The get_fanouts command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the [trace_flat_model](#) command to perform tracing on the flat model with simulation conditions considered.

Arguments

- *obj_spec*

A required value that specifies a Tcl list of one or more net, pin, port, or pseudo_port names; or a collection of one or more net, pin, port, or pseudo_port objects.

- -stop_on cell_pin | net | pin

An optional switch and literal pair that specifies the following:

- When -stop_on is set to cell_pin (the default), the fanout walk traverses nets and design pins but stops on pins of cells or primitives in the fanout of the object. The

walk may still, however, stop on a net or a design pin when the net or design pin does not have a fanout.

- When -stop_on is set to net, the fanout walk traverses through one level of “assign” statements and stops on the net connected to the start pin.,
- When -stop_on is set to pin, the fanout walk traverses nets and reports the pin in the fanout of the object. This could be output or inout pins above the object or inputs or inout pins on child instances. The walk may still, however, stop on a net if the net does not fanout to pins.
- -silent

An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Examples

Example 1

The following example shows a series of get_fanouts command invocations of the circuit example shown in [Figure 4-4](#).

```
#Find cell pins in the fanout of u4/I5
get_fanouts u4/I5
{u4/u4/A2 u4/u5/u3/D}

#Find pins in the fanout of u4/u1/Y. Returns a cell pin as there is no design pin before it
get_fanouts u4/u1/Y -stop_on pin
{u4/u2/D}

#Find pins in the fanout of u4/u4/Y
get_fanouts u4/u4/Y -stop_on pin
{u4/u1/A2 u4/u5/I2}

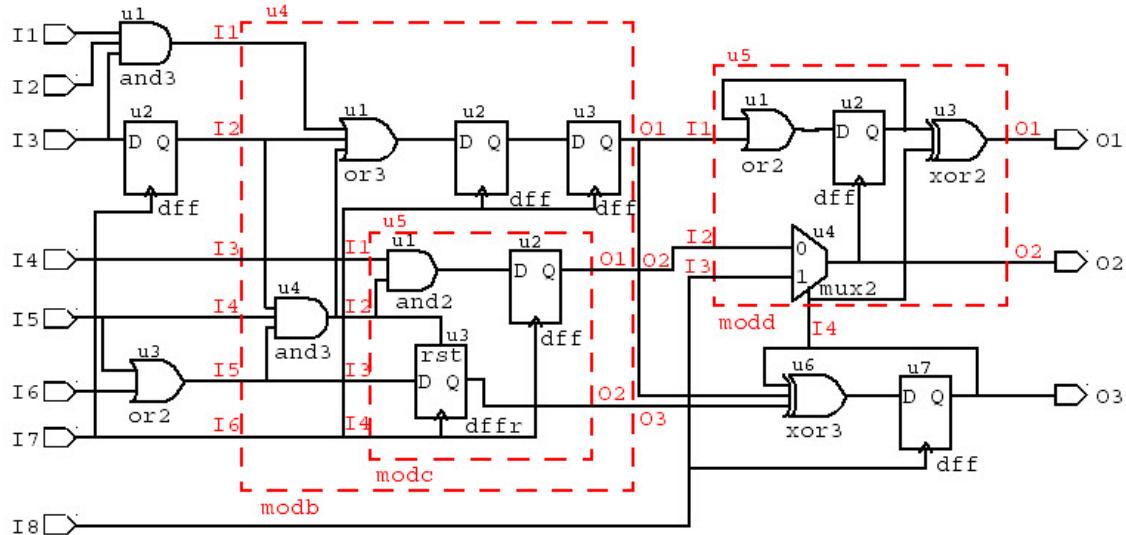
#Find all nets connected to the Q pin of dff modules
get_fanouts [get_pins -filter {module_name == "dff" && leaf_name == "Q"}] -stop_on net
{n2 O3 u4/n3 u4/O1 u4/u5/O1 u4/u5/O2 u5/n2}

#Find net in fanout of a cell output pin
get_fanouts u1/Y -stop_on net
{n1}
```

Example 2

The following example adds an internal primary output (pseudo_port). Based on the example circuit shown in [Figure 4-4](#), it then returns the collection of objects found in the fanout of the pseudo_port’s underlying pin.

```
add_primary_outputs {u4/u5} -internal
get_fanouts [get_ports u4/u5 -of_type pseudo]
{u4/u4/A2 u4/u5/u3/D}
```

Figure 4-4. Circuit Illustrating the Usage of the get_fanouts Command**Related Topics**

[get_fanins](#)
[get_icl_ports](#)
[get_nets](#)

[get_fault_type](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: all modes

Returns the current fault model type, as previously specified by the [set_fault_type](#) command.

Usage

```
get_fault_type
```

Description

Returns the current fault model type, as previously specified by the [set_fault_type](#) command.

The command returns one of the following strings: stuck, toggle, iddq, transition, path_delay, bridge, udfm.

Arguments

None

Examples

Example 1

The following example shows how to set the fault model type and then introspect the fault model:

```
ANALYSIS> set_fault_type bridge  
ANALYSIS> get_fault_type  
  
bridge
```

Example 2

The following example loads an SDC file if the current fault model type is “transition”:

```
if {[get_fault_type] eq "transition" } {  
    read_sdc myDesign.sdc  
} else {  
    puts "No SDC file read. Fault type is [get_fault_type]."  
}
```

Related Topics

[set_fault_type](#)

get_gate_pins

Context: all contexts

Mode: setup, analysis

Prerequisites: Although you can use this command in all system modes, you can only use it with a flattened netlist. In Tessent Shell, netlist flattening typically happens when you first attempt to enter analysis mode or when you issue the `create_flat_model` command.

Returns a collection of `gate_pin` objects found below the current design in the flat model.

Usage

```
get_gate_pins [gate_pin_id_list | hier_pin_name_patterns] [-filter attribute_equation] [-regexp]
[-nocase] [-silent] [-of_instances instance_objects] [-below_instances instance_objects] [-of_pins pin_objects] [-of_ports port_objects] [-of_nets net_objects] [-of_gates {gate_id_list |
gate_pin_objects} | -from_objects from_objects]} [-of_gate_type {sequential / primitive_name_list}]
```

Description

Returns a collection of `gate_pin` objects found below the current design in the flat model.

The command supports arbitrary filtering based on attribute values. It also supports those options for frequently used filters.

Arguments

- *gate_pin_id_list* | *hier_pin_name_patterns*

An optional string that specifies a list of `gate_pin` objects. The *gate_pin_id_list* is a list of `gate_pin` IDs (two integers separated by a period, the first integer representing a gate and the second representing a `gate_pin`). No wildcards are allowed in the *gate_pin_id_list*. If the second integer is not present, the ID specifies all pins on the gate.

The *hier_pin_name_patterns* is a list of strings (wildcard patterns or regular expressions) matching hierarchical pin names. Using the *hier_pin_name_patterns* is equivalent to filtering on the `hier_pin_name` attribute.

Note

 When the `get_gate_pins` command operates on the flat model, it removes all backslashes and white spaces from escaped identifiers. You must not include backslashes or white spaces in the *hier_pin_name_patterns* string. If you want to access the corresponding `gate_pin`, use the `gate_pin_id` attribute of the hierarchical pin object. If you want to access the `gate_pin` by name, use the [get_name_list -remove_escaping](#) command and switch.

- -filter *attribute_equation*

An optional switch and string that specify to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the

gate_pin object type. See “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The `=~` and `!~` matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the `-regexp` option or not.

- **-regexp**

An optional switch that directs the tool to interpret the value of the `hier_pin_name_patterns` argument as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for gate_pins matching the value of the `hier_pin_name_patterns` argument.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the `hier_pin_name_patterns` disappeared due to a change in a previous editing step. This `-silent` option has no effect on any other error reporting.

Note

You can use the `-silent` switch to manage the case in which an empty collection is returned. Before using the `-silent` switch for this purpose, refer to the `set_tcl_shell_options -change_no_result_warnings_to_errors` switch description for detailed usage.

- **-of_instances *instance_objects***

An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *instance_objects*.

- **-below_instances *instance_objects***

An optional switch and value pair that constrains the command to return only gate_pins associated with the circuitry located underneath the specified *instance_objects*.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the `-below_instances` switch of the [get_instances](#) command description for the justification of this behavior.

- **-of_pins *pin_objects***

An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *pin_objects*.

- **-of_ports *port_objects***

An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *port_objects* or *pseudo_port* objects.

- **-of_nets *net_objects***

An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *net_objects*. If a gate_pin name directly matches the net name, the matching gate_pin is returned; otherwise, the driving gate_pin object of the net is returned.

- **-of_gates {*gate_id_list* / *gate_pin_objects*}**

An optional switch and value pair that constrains the command to return only gate_pins associated with the specified gates. The specified gates can be a list of gate IDs, gate_pin IDs, gate_pin objects or hierarchical pin name patterns.

- **-from_objects *from_objects***

An optional switch and value pair that constrains the command to return only objects specified with the *from_objects* parameter, which may contain ports, pseudo_ports, pins, nets, and gate_pins. A port, pseudo_port, pin or net is mapped to the corresponding gate_pins in the same way the existing -of_ports, -of_pins or -of_nets options do. A gate_pin is just passed through to the result collection. This new option is convenient if you have a collection of design objects of different types that you want to map to gate_pins.

- **-of_gate_type {*sequential* / *primitive_name_list*}**

An optional switch and value pair that constrains the command to return only gate_pins associated with the gates of the specified types.

When the gate type *sequential* is specified, the command returns all the gate_pin objects associated with the gates whose primitive_name attribute value is DFF, DLAT, RAM or ROM .The allowed values for the *primitive_name_list* are all the primitive_name attribute values (see [Table 8-1, Built-In Primitives and Primitive Port Names](#)). The value of *primitive_name_list* can be either a single string or a Tcl list of strings such as {PI PO}.

Examples

Example 1

The following example filters the collection of all gate_pin objects based on an attribute_equation of user-defined attributes:

```
get_gate_pins -filter {myType1 || myType2 == "ABC"}
{/u1/U2/b 16.0 16.1 16.2 24.0}
```

Example 2

The following example returns the collection of all gate_pin objects matching a list of case-insensitive hierarchical pin name patterns. For more examples of pattern matching with regular expressions, refer to “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164.

```
get_gate_pins {u1/u1/a u*/b} -nocase
{/u1/U1/a /u1/U1/b /u1/U2/b /u1/U3/b /u2/U1/b}
```

Example 3

The following example returns a collection of all gate_pin objects on the gate with ID=12, the output gate_pin object (pin_index=0) on the gate with ID=14, and the gate_pin object with pin_index=2 on the gate with ID=11:

```
get_gate_pins { 12 14.0 11.2 }
{/u2/U1/b 12.0 14.0 /u1/U3/c}
```

Example 4

The following equivalent get_gate_pins commands return a collection of all gate_pin objects where the corresponding hierarchical pin name matches the wildcard pattern /u1/*/a or /u2/U1*:

```
get_gate_pins { /u1/*/a /u2/U1* }
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}
get_gate_pins -filter "hier_pin_name =~ /u1/*/a || hier_pin_name =~ /u2/U1*"
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}
```

Example 5

The following example adds to the fault list all of the gate_pin objects corresponding to the hierarchical pins matching the wildcard pattern u1/*:

```
add_faults [get_gate_pins u1/*]
report_faults
```

type	code	pin_pathname
0	UC	/u2/U1/b
1	UC	/u2/U1/b
0	UC	/u2/U1/c
1	UC	/u2/U1/c
0	UC	/u2/U1/a
1	UC	/u2/U1/a
0	UC	/u2/U1/y
1	UC	/u2/U1/y
0	UC	/u2/U1/z
1	UC	/u2/U1/z

Example 6

The following examples are based on the circuit in [Figure 4-3](#) on [page 847](#). This example returns all of the gate pins for the specified instance u4/u2:

```
SETUP> get_gate_pins -of_instances u4/u2
{u4/u2/Q u4/u2/QB u4/u2/CLK u4/u2/D}
```

This example returns all of the gate pins named D that are found below the u4 level of hierarchy:

```
SETUP> get_gate_pins */D -below_instances u4
```

```
{u4/u2/D u4/u3/D u4/u5/u2/D u4/u5/u3/D}
```

Example 7

The following examples return all the gate pins in gates of type DFF:

```
get_gate_pins --filter "primitive_name == DFF"
```

```
{7.0 7.1 7.2 DFF/CLK DFF/D}
```

```
get_gate_pins --of_gate_type DFF
```

```
{7.0 7.1 7.2 DFF/CLK DFF/D}
```

This example returns gate pins named D in all gates of type DFF:

```
get_gate_pins -of_gate_type DFF -filter "primitive_port_name == D"
```

```
{DFF/D}
```

Related Topics[get_fanins](#)[get_common_parent_instance](#)[get_modules](#)[get_nets](#)

[get_icl_extraction_options](#)

Context: all contexts

Mode: all modes

Provides access to the settings specified by the [set_icl_extraction_options](#) command.

Usage

```
get_icl_extraction_options -extract_learned_muxes | -extract_library_muxes
```

Description

Provides access to the settings specified by the [set_icl_extraction_options](#) command.

Arguments

- `-extract_learned_muxes`

A switch that instructs the tool to return the `extract_learned_muxes` setting of the ICL extraction options. Possible values are 0 and 1.

- `-extract_library_muxes`

A switch that instructs the tool to return the `extract_library_muxes` setting of the ICL extraction options. Possible values are 0 and 1.

Examples

The following command returns the setting specified by the `-extract_library_muxes` option of [set_icl_extraction_options](#) command:

```
get_icl_extraction_options -extract_library_muxes
```

1

Related Topics

[set_icl_extraction_options](#)

[report_icl_extraction_options](#)

get_icl_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fanin of the specified pin or port objects.

Usage

```
get_icl_fanins obj_spec [-stop_on none | pin] [-silent]
```

Description

Returns a collection of all requested objects found in the fanin of the specified pin or port objects.

The `-stop_on` option is used to specify object types that the tracing should stop on. Specifying this option does not mean that the returned collections will only contain objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be directly connected to a port. The port will be reported as the object on the fanin of the pin even if `-stop_on` is specified as pin.

Arguments

- ***obj_spec***

A required value that specifies a Tcl list of one or more ICL pin or port names; or a collection of one or more ICL pin or port objects.

- **-stop_on none | pin**

An optional switch and literal pair that specifies the following:

none — The fanin walk traverses the connections and the pins in the fanin of the object.
The trace reports pins with no fanin or ports reached in the fanin walk.

pin — The fanin walk traverses the connection and reports the pin in the fanin of the object; this could be an input pin above the object or an output pin on child instances.
The walk may still, however, stop on a port if the pin does not fanin to pins.

- **-silent**

An optional switch that suppresses the error that is normally generated if any object within *obj_spec* does not exist, in which case it is ignored.

Examples

The following example shows command invocations with and without the `-stop_on pin` option to illustrate its effect.

```
# find true fanin of pin block1.sib1.si
```

```
get_icl_fanins block1.sib1.si
{si}

# Went through block1.si as -stop_on was defaulted to none.
# Reached ICL port si on the current_design find first pin in fanin of
# block1.sib1.si

get_icl_fanins block1.sib1.si -stop_on pin
{block1.si}

# Stopped on block1.si as -stop_on was specified as pin.
```

Related Topics

[get_icl_ports](#)

get_icl_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fanout of the specified ICL pin, or port objects.

Usage

```
get_icl_fanouts obj_spec [-stop_on none | pin] [-silent]
```

Description

Returns a collection of all requested objects found in the fanout of the specified ICL pin, or port objects.

The -stop_on option is used to specify object types that the tracing should stop on. Specifying this option does not mean that the returned collections will only contain objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be directly connected to a port. The port will be reported as the object on the fanout of the pin even if -stop_on is specified as pin.

Arguments

- *obj_spec*

A required value that specifies a Tcl list of one or more ICL pin or port names; or a collection of one or more ICL pin or port objects.

- -stop_on none | pin

An optional switch and literal pair that specifies the following:

none — The fanout walk traverses the connections and the pins in the fanout of the object. The trace will report pins with no fanout or ports that were reached in the fanout walk.

pin — The fanout walk traverses the connection and reports the pin in the fanout of the object. This could be an output pin above the object or an input pin on child instances. The walk may still, however, stop on a port if the pin does not fanout to pins.

- -silent

An optional switch that suppresses the error that is normally generated if any object within *obj_spec* does not exist, in which case it is ignored.

Examples

The following example shows the get_icl_fanouts command invocations with and without the -stop_on pin option to illustrate its effect.

```
# find true fanout of pin block1.sib1.so
```

```
get_icl_fanouts block1.sib1.so
{so}

# Went through block1.so as -stop_on was defaulted to none. Reached ICL
# port so on the current_design find first pin in fanout of block1.sib1.so

get_icl_fanouts block1.sib1.so -stop_on pin
{block1.so}

# Stopped on block1.so as -stop_on was specified as pin.
```

Related Topics

[get_fanins](#)

[get_icl_ports](#)

get_icl_instances

Context: all contexts

Mode: all modes

Returns a collection of all ICL instances instantiated relative to the current design that match the specified *name_patterns list or -filter expression.

Usage

Usage 1: Without the -hierarchical option

```
get_icl_instances [name_patterns] [-of_modules obj_spec]
  [-parent_of_instances instance_objects | -below_instances instance_objects |
   -of_pins icl_pin_objects]
  [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Usage 2: With the -hierarchical option

```
get_icl_instances [leaf_name_patterns] [-of_modules obj_spec]
  [-parent_of_instances instance_objects | -below_instances instance_objects]
  [-hierarchical] [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Usage 3: With no name pattern

```
get_icl_instances [-of_modules obj_spec]
  [-parent_of_instances instance_objects | -below_instances instance_objects |
   -of_pins icl_pin_objects]
  [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all ICL instances instantiated relative to the current design that match the specified *name_patterns list or -filter expression.

Usage 1: The name patterns are expressed as hierarchical instance path names relative to the current design. You can have any number of wildcards or regular expressions in the names between the hierarchy separator symbols as shown in the examples for this command. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

Usage 2: The name patterns are leaf instance names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf instance name pattern is searched relative to all ICL instances found below the current design.

Usage 3: A name_patterns argument is not specified so that all ICL instances are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all ICL instances that match the results returned by the -filter attribute_equation filter as shown in Usage 3. In Usage 2, described previously, the name patterns cannot include slashes or dots which are hierarchy separator symbols. In Usage 1, you can use the dot or slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

- *-of_modules obj_spec*

An optional switch and value pair that constrains the command to return only instances of ICL modules specified by obj_spec. The obj_spec is a Tcl list of one or more ICL module names, or a collection of one or more ICL module objects as returned by the [get_icl_modules](#) command.

- *-parent_of_instances instance_objects*

An optional switch and value pair that constrains the command to return a collection of ICL instance objects that are the parent instances of the specified instance_objects. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional name_patterns option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -parent_of_instances option is used.

- *-below_instances instance_objects*

An optional switch and value pair that constrains the command to search for ICL instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

If the specified instance_spec is an empty collection, all instances below the current design are considered. See the description of the [-below_instances switch](#) of the [get_instances](#) command description for the justification of this behavior.

- *-of_pins icl_pin_objects*

An optional switch and value pair that constrains the command to return the ICL instance to which the ICL pins belong. The returned collection is filtered to return any given ICL instance only once, even if the instance belongs to many ICL pins in the icl_pin_objects list. When this option is specified, the optional name_patterns must be leaf_name patterns that are checked against the leaf name of the ICL instances. An error is generated if the specified name_patterns contains more than one level of hierarchy when the -of_pins option is used.

- **-hierarchical**
An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the name_patterns argument cannot include slashes or dot which are hierarchy separator symbols.
- **-filter *attribute_equation***
An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL instance object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on the filtering attribute equation format.
- **-regexp**
An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. You must properly escape the bracket “[]” symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax. You must use the slash as a hierarchy separator when using the -regexp option.
- **-nocase**
An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the name_patterns.
- **-silent**
An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the name_patterns disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example illustrates Usage 3 and returns the collection of all instances of ICL modules modb and modd found anywhere below the current design:

```
get_icl_instances -of_modules {modb modd}
{u2 u1/u5}
```

Example 2

The following example illustrates Usage 3 in which the collection of all instances is filtered based on an attribute_equation of user-defined attributes.

```
get_icl_instances -filter {myType1 || myType2 == "ABC"}
{u2 u1.u5}
```

Example 3

The following example illustrates Usage 1 in which the collection of all instances matching a list of hierarchical name patterns is returned. The names u1/* and u*/u* are only searched relative to the current design.

```
get_icl_instances {u1/* u*/u*}
{u4.u5 u1.u7}
```

Example 4

The following example illustrates Usage 2 in which the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

```
get_icl_instances u3* -hierarchical
{u3 u4.u30 u4.u5.u31 u5.u3}
```

Related Topics

[get_icl_modules](#)
[get_icl_ports](#)
[read_icl](#)

get_icl_modules

Context: all contexts

Mode: all modes

Returns a collection of all ICL modules that match the specified *name_patterns* list or -filter expression.

Usage

```
get_icl_modules [name_patterns] [-of_instances instance_objects |  
-below_instances instance_objects | -parameterized_views_of_modules  
icl_module_objects] [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all ICL modules that match the specified *name_patterns* list or -filter expression.

By default, this command returns all ICL modules below the current design. You can use the -of_instances argument to constrain the search.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL modules. If no *name_patterns* are specified, the tool searches all ICL modules that match the results returned by the -filter *attribute_equation* filter.

- -of_instances *instance_objects*

An optional switch and value pair that constrains the command to return only ICL modules of the specified *instance_objects*. The *instance_objects* is a Tcl list of one or more ICL instance names, or a collection of one or more ICL instance objects as returned by the get_icl_instances command. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient.

- -below_instances *instance_objects*

An optional switch and value pair that constrains the command to return only ICL modules corresponding to instances found below the specified *instance_objects*.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the [-below_instances switch](#) of the [get_instances](#) command description for the justification of this behavior.

- **-parameterized_views_of_modules *icl_module_objects***

An optional switch and value pair that constrains the command to return a collection of ICL modules corresponding to all the parameterized views of the ICL module objects specified. The *icl_module_objects* can be specified as a Tcl list of ICL module names or a collection of ICL modules. Whether *icl_module_objects* contains a parameterized view or the master copy of an ICL module, this switch instructs the command to find all parameterized views and return them. Using this switch is equivalent to using the *master_name* attribute of an ICL module and requesting all ICL modules having the same *master_name* value. Using this switch, however, is much more efficient. You use this switch to determine whether an ICL module is uniquely instantiated or not and to find out if an ICL module has multiple parameterized views.

- **-filter *attribute_equation***

An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the ICL module object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The `=~` and `!~` matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the `-regexp` option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the `=~` and `!~` -filter expressions as regular expressions instead of as simple wildcard patterns. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This `-silent` option has no effect on any other error reporting.

Note

 You can use the `-silent` switch to manage the case in which an empty collection is returned. Before using the `-silent` switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

The following example returns ICL modules whose name matches a* or b*.

```
get_icl_modules {a* b*}
{ab abc b bcd}
```

Example 2

The following example returns ICL modules having one or two characters.

```
get_icl_modules {[alnum]{1,2}} -regexp
{ab b}
```

Example 3

This example shows how to report the names of the parameterized views for a module.

```
puts “[get_name_list [get_icl_modules -parameterized_views_of_modules block]]”
block block@PV1 block@PV2
```

Related Topics

[get_icl_instances](#)
[get_icl_ports](#)
[read_icl](#)

get_icl_module_parameter_list

Context: all contexts

Mode: all modes

Returns the list of parameters in an ICL module.

Usage

```
get_icl_module_parameter_list module_object [-overridden]
```

Description

Returns the list of parameters in an ICL module.

Arguments

- ***module_object***

A required string that specifies the parameterized ICL module objects whose parameters are to be returned.

- -overridden

An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Values

List of parameters.

Examples

In the following example, this command is used to obtain the list of parameters overridden in the specific instantiation. Notice how the `get_icl_instances` command is used to find the ICL instance u1/myinst_i1; this ensures that the `get_attribute_value_list` command returns the `module_name` attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]  
set params [get_icl_module_parameter_list [get_attribute_value_list $icl_inst \  
-name module_name] -overridden]
```

Related Topics

[get_icl_module_parameter_value](#)

get_icl_module_parameter_value

Context: all contexts

Mode: all modes

Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.

Usage

```
get_icl_module_parameter_value parameter_name [-of_module icl_module_object]
```

Description

Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.

Arguments

- *parameter_name*
A required string that specifies the name of the parameter whose value is to be returned.
- -of_module *icl_module_object*
An optional switch and string pair that specifies the ICL module whose parameter value is to be returned. When omitted, the parameters of the current design are introspected.

Return Values

The value of the requested ICL module parameter.

Examples

In the following example, the values of all overridden parameters are echoed in the transcript. Notice how the [get_icl_instances](#) command is used to find the ICL instance u1/myinst_i1; this ensures that the [get_attribute_value_list](#) command returns the module_name attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]
set icl_mod [get_attribute_value_list $icl_inst -name module_name]
set params [get_icl_module_parameter_list $icl_mod -overridden]
foreach param $params {
    puts "parameter $param = [get_icl_module_parameter_value $param \
        -of_mod $icl_mod]"
}
```

Related Topics

[get_icl_module_parameter_list](#)

get_icl_objects

Context: all contexts

Mode: all modes

Returns a collection of ICL object matching the requested criteria.

Usage

```
get_icl_objects [name_patterns] -object_types object_types
    [-below_instances icl_instance_spec | -parent_of_instances icl_instance_spec
     | -in_modules icl_module_spec | -of_objects icl_object_specs]
    [-hierarchical]
    [-direction direction]
    [-function function]
    [-filter attribute_equation]
    [-regexp]
    [-nocase]
    [-silent]
```

Description

Returns a collection of ICL object matching the requested criteria.

This command combines the complete usage of [get_icl_modules](#), [get_icl_instances](#), [get_icl_ports](#), and [get_icl_pins](#) plus provides introspection into other ICL object types as well as the ability to see objects inside a module before it is elaborated. [Table 4-5](#) lists the available object types and against which object they can be requested from. The object types `icl_instance`, `icl_pin`, `icl_scan_interface`, `icl_scan_register`, `icl_scan_mux` and `icl_one_hot_scan_group` are only available when ICL is elaborated. You can get access to their `_of_module/_in_module` counterpart without requiring the ICL to be elaborated, but you are limited to the object directly declared in the ICL module. This difference is illustrated in [Example 2](#) below.

Arguments

- *name_patterns*
An optional Tcl list of one or more strings that specifies patterns to be used to filter the returned list of ICL object based on their name attributes. If no *name_patterns* are specified, the tool searches all ICL object that matches the other criteria.
- **-object_types object_types**
A required Tcl list of one or more strings that specifies ICL object type to consider in the search. The currently supported list of ICL object types are the following:
 - `icl_module`
 - `icl_port`
 - `icl_instance`

- `icl_instance_in_module`
- `icl_pin`
- `icl_scan_interface`
- `icl_scan_interface_of_module`
- `icl_scan_register`
- `icl_scan_register_in_module`
- `icl_scan_mux`
- `icl_scan_mux_in_module`
- `icl_one_hot_scan_group`
- `icl_one_hot_scan_group_in_module`
- `icl_alias`
- `icl_alias_in_module`
- `-below_instances icl_instance_spec`

An optional switch-value pair used to specify a Tcl list of one or more names of `icl_instance` objects or a collection containing one or more `icl_instance` objects. This option can only be used when the `-object_type` switch specifies those object type:

- `icl_instance`
- `icl_pin`
- `icl_scan_interface`
- `icl_scan_register`
- `icl_scan_mux`
- `icl_one_hot_scan_group`
- `icl_alias`

If the specified `instance_spec` is an empty collection, all instances below the current design are considered. See the description of the `-below_instances` switch of the [get_instances](#) command description for the justification of this behavior

- `-parent_of_instances icl_instance_spec`

An optional switch-value pair used to specify a Tcl list of one or more names of `icl_instance` objects or a collection containing one or more `icl_instance` objects. This option can only be used when the `-object_type` switch specifies “`icl_instance`”.

- **-in_modules *icl_module_spec***

An optional switch-value pair used to specify a Tcl list of one or more names of ICL modules or a collection contain one or more ICL module objects. The allowed types in the **-object_types** switch value is “*icl_instance_in_module*” and “*icl_scan_register_in_module*” when the **-in_modules** switch is used.

- **-of_objects *icl_object_spec***

An optional switch-value pair used to specify a Tcl list of one or more names of ICL objects or a collection contain one or more ICL objects. Table 4-5 shows the allowed type for the *icl_object_spec* objects based on the **-object_types** requested.

Table 4-5. ICI objects and which options they apply to

icl_object types in the -of_objects <i>icl_object_spec</i>	allowed types in the -object_types list
<i>icl_module</i>	<i>icl_port</i> <i>icl_instance</i> <i>icl_scan_interface_of_module</i>
<i>icl_instance</i>	<i>icl_module</i> <i>icl_pin</i> <i>icl_scan_interface</i> <i>icl_scan_register</i> <i>icl_scan_mux</i> <i>icl_one_hot_scan_group</i> <i>icl_alias</i>
<i>icl_instance_in_module</i>	<i>icl_module</i>
<i>icl_pin</i>	<i>icl_instance</i> <i>icl_scan_interface</i>
<i>icl_scan_interface</i>	<i>icl_pin</i> <i>icl_instance</i>
<i>icl_scan_interface_of_module</i>	<i>icl_module</i> <i>icl_port</i>
<i>icl_scan_register</i>	<i>icl_instance</i>
<i>icl_scan_register_in_module</i>	<i>icl_module</i>
<i>icl_scan_mux</i>	<i>icl_instance</i>
<i>icl_scan_mux_in_module</i>	<i>icl_module</i>
<i>icl_one_hot_scan_group</i>	<i>icl_instance</i>
<i>icl_one_hot_scan_group_in_module</i>	<i>icl_module</i>

Table 4-5. ICL objects and which options they apply to (cont.)

icl_object types in the -of_objects icl_object_spec	allowed types in the -object_types list
icl_alias	icl_instance
icl_alias_in_module	icl_module

- **-hierarchical**
An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the *name_patterns* argument cannot include more than one dot which is the hierarchy separator symbols in ICL.
- **-direction direction**
An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the ICL pin or port objects. When omitted, pins or ports with any direction are considered.
- **-function function_list**
An optional switch-value pair that specifies to filter results based on the value of the ijttag_function attribute on the ICL pin or port object. When omitted, ports or pins with any ijttag_function value are considered. The ijttag_function values are the ICL port function as found in the ICL description where multiple words are separated by underscores instead of using camel casing without the “_port” termination. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.
- **-filter attribute_equation**
An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for all the object types listed in the -object_type switch. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.
- **-regexp**
An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard pattern.
- **-nocase**
An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ICL object names matching *name_patterns*.
- **-silent**
An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the

name_patterns disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the `set_tcl_shell_options -change_no_result_warnings_to_errors` switch description for detailed usage.

Examples

Example 1

The following example illustrate how you can query objects of multiple types using a single invocation of the `get_icl_objects` command. This cannot be done when using `get_icl_ports` and `get_icl_pins` command. The example usage queries for ICL ports and ICL pins with `ijtag_function` equal to `capture_en` within the elaborate ICL design.

```
get_icl_objects -object_type {icl_pin icl_port} -function capture_en
{corea_rtl1_tessent_mbist_bap_inst.capture_en
 corea_rtl1_tessent_sib_mbist_inst.ijtag_ce
 corea_rtl1_tessent_sib_sti_inst.ijtag_ce
 corea_rtl2_tessent_edt_c1_tdr_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_clkb_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_clkb_inst.tdr_sib.capture_en
 corea_rtl2_tessent_occ_child_nocg_clka_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_nocg_clka_inst.tdr_sib.capture_en
 corea_rtl2_tessent_occ_child_nocg_clkc_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_nocg_clkc_inst.tdr_sib.capture_en
 corea_rtl2_tessent_sib_edt_inst.ijtag_ce
 corea_rtl2_tessent_sib_occ_inst.ijtag_ce
 corea_rtl2_tessent_sib_sri_inst.ijtag_ce
 corea_rtl2_tessent_tdr_sri_ctrl_inst.ijtag_ce ijtag_ce}
```

Example 2

The following example illustrates the difference between the object types ending with “_of_module” vs not. The “`icl_scan_interface`” object_type is a hierarchical object_type which exists on the current design and on ICL instances. Their names are the full hierarchical paths of their associated ICL instances follow by the leaf_name of the scan interface. They only exist when the ICL is elaborated. The “`icl_scan_interface_of_module`” object_type is a non-hierarchical object_type which exist on ICL modules. Their name are the leaf_name of the scan interface. They exist even if ICL is not elaborated. The top part of the example uses the hierarchical object types to find specific instances of an ICL scan interface and follows to extract the ICL pins with function “`scan_in`” from it. The second part of the example uses the non-hierarchical object types to find the ICL scan interface of a module and follows to extract the ICL port with function “`scan_in`” from it. The second method is useful when you want to extract information about ICL modules and the top level ICL is not yet extracted and elaborated. It is also useful to introspect ICL modules not part of the current ICL elaboration tree.

This part shows introspecting the ICL pins of ICL scan interfaces when ICL is elaborated

```
set icl_scan_int [get_icl_objects -object_type icl_scan_interface \
    -of_objects [get_icl_instances corea_rtl2_tessent_sib_sri_inst]]
{corea_rtl2_tessent_sib_sri_inst.client
 corea_rtl2_tessent_sib_sri_inst.host}

set icl_pins [get_icl_objects -object_type icl_pin \
    -of_object $icl_scan_int \
    -function scan_in]
{corea_rtl2_tessent_sib_sri_inst.ijtag_si
 corea_rtl2_tessent_sib_sri_inst.ijtag_from_so}
```

This part shows introspecting the ICL pins of ICL scan interfaces when ICL is elaborated

```
set icl_scan_int [get_icl_objects -object_type icl_scan_interface \
    -of_objects [get_icl_instances corea_rtl2_tessent_sib_sri_inst]]
{corea_rtl2_tessent_sib_sri_inst.client
 corea_rtl2_tessent_sib_sri_inst.host}

set icl_pins [get_icl_objects -object_type icl_pin \
    -of_object $icl_scan_int \
    -function scan_in]
{corea_rtl2_tessent_sib_sri_inst.ijtag_si
 corea_rtl2_tessent_sib_sri_inst.ijtag_from_so}
```

This part shows introspecting the ICL ports of ICL scan interfaces of ICL modules when # ICL is NOT elaborated

```
set icl_scan_int_of_mod [get_icl_objects -object_type icl_scan_interface_of_module \
    -of_objects [get_icl_modules corea_rtl2_tessent_sib_1]]
{client host}

set icl_port [get_icl_objects -object_type icl_port -of_objects $icl_scan_int_of_mod \
    -function scan_in]
{ijtag_si ijtag_from_so}
```

Related Topics

[get_common_parent_instance](#)

[get_icl_ports](#)

get_icl_pins

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified *name_patterns* list or -filter expression.

Usage

Usage 1: Without the -hierarchical option

```
get_icl_pins name_patterns [-direction input | output]
[-function function]
[-of_instances instance_objects | -below_instances instance_objects]
[-filter attribute_equation]
[-regexp] [-nocase] [-silent]
```

Usage 2: With the -hierarchical option

```
get_icl_pins leaf_name_patterns | leaf_instance/leaf_name_patterns -hierarchical
[-below_instances instance_objects] [-filter attribute_equation]
[-direction input | output] [-function function]
[-regexp] [-nocase] [-silent]
```

Usage 3: With no name pattern

```
get_icl_pins [-filter attribute_equation] [-direction input | output] [-function function] [-regexp]
[-nocase] [-silent]
```

Description

Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified *name_patterns* list or -filter expression.

Usage 1: The name patterns are expressed as hierarchical pin pathnames relative to the current design. You can have any number of wildcards or regular expressions in the names between the hierarchy separator symbols as shown in the examples that follow. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

Usage 2: The name patterns are leaf pin names or leaf instance name/leaf pin names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf name pattern is searched relative to all instances found below the current design. There can only be one hierarchy separator when using the -hierarchical option as shown in Example 3.

Usage 3: A *name_patterns* argument is not specified so all ICL pins are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns / leaf_instance/leaf_name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all ICL pins that match the results returned by the *-filter attribute_equation* filter as shown in Usage 3. In Usage 2, described previously, the name patterns are not allowed to include more than one slash or dot which are hierarchy separator symbols. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for a complete description of the pattern syntax when the *-regexp* option is used or not.

- *-direction input | output*

An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- *-of_instances instance_objects*

An optional switch and value pair that constrains the command to return only ICL pins of the specified *instance_objects*. The *instance_objects* is a Tcl list of one or more ICL instance names, or a collection of one or more ICL instance objects as returned by the *get_icl_instances* command. You could use the *module_name* attribute of the *instance_objects* to perform a similar filtering but this method is more direct and efficient.

- *-below_instances instance_objects*

An optional switch and value pair that constrains the command to search for ICL pins below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the *-below_instances* switch of the [get_instances](#) command description for the justification of this behavior.

- *-hierarchical*

An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the *name_patterns* argument cannot include more than one slash or dot which are the hierarchy separator symbols.

- *-function function*

An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL pin object. When omitted, pins with any function are considered. The value function is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- **-filter *attribute_equation***

An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the ICL pin object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The $=\sim$ and $!\sim$ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the *-regexp* option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the $=\sim$ and $!\sim$ -filter expressions as regular expressions instead of as simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ICL pin names matching *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This *-silent* option has no effect on any other error reporting.

Note

 You can use the *-silent* switch to manage the case in which an empty collection is returned. Before using the *-silent* switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

The following example illustrates Usage 3 where the collection of all ICL pins is filtered based on an *attribute_equation* of user-defined attributes.

```
get_icl_pins -filter {myType1 || myType2 == "ABC"}  
{u1.Y u4.u4.A0 u7.D}
```

Example 2

The following example illustrates Usage 1 where the collection of all ICL pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for examples of pattern matching with regular expressions.

```
get_icl_pins {u1/Y u*/Q}
{u1.Y u2.Q u7.Q}
```

Example 3

The following example illustrates Usage 2 where the collection of all ICL pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all ICL instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

```
get_icl_pins u*/Y* -hierarchical
{u1.Y u3.Y u4.u1.Y u4.u4.Y u4.u5.u1.Y u5.u1.Y u5.u3.Y u5.u4.Y u6.Y}
```

Example 4

The following example gets all ICL pins with function clock or to_clock in the ICL data base.

```
get_icl_pins -function {clock to_clock}
{PLL1.ref PLL1.vco mbist.clk}
```

Related Topics

[get_common_parent_instance](#)

[get_icl_ports](#)

[read_icl](#)

get_icl_ports

Context: all contexts

Mode: all modes

Returns a collection of all ports on a given module that match the specified *name_patterns* list or -filter expression.

Usage

```
get_icl_ports [name_patterns] [-of_modules obj_spec] [-direction input | output]  
[-function function] [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all ports on a given module that match the specified *name_patterns* list or -filter expression.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ports. If no *name_patterns* are specified, the tool searches all ports that match the results returned by the -filter *attribute_equation* filter, if one is specified.

- -of_modules *obj_spec*

An optional switch and value pair that constrains the command to return only ports on modules specified by *obj_spec*. The *obj_spec* is a Tcl list of ICL module names or a collection of one or more module objects as returned by the get_icl_modules command.

- -direction input | output

An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- -function *function*

An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL port object. When omitted, ports with any function are considered. The value function is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- **-filter *attribute_equation***

An optional switch and string that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the ICL port object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The $=\sim$ and $!\sim$ matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the **-regexp** option or not.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the $=\sim$ and $!\sim$ -filter expressions as regular expressions instead of as simple wildcard patterns. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This **-silent** option has no effect on any other error reporting.

Note

 You can use the **-silent** switch to manage the case in which an empty collection is returned. Before using the **-silent** switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

This example looks for ICL ports on the current design whose name match I* or O*.

```
get_icl_ports {I* O*}
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

```
get_icl_ports {[[:alnum:]]{1,2}} -regexp -of_module ModA
{ab a b}
```

Related Topics

[get_common_parent_instance](#)

[read_icl](#)

get_icl_scan_interface_list

Context: all contexts

Mode: all modes

Returns the *names of the* scan interfaces in an existing ICL top module, if the ICL top module exists.

Usage

```
get_icl_scan_interface_list [ -control_of object_spec ]
```

Description

Returns the *names of the* scan interfaces in an existing ICL top module, if the ICL top module exists. If the ICL top module does not exist, it returns the names of the scan interfaces that will be created in the new ICL top module by ICL extraction after the `add_icl_scan_interfaces` command has been used to specify the scan interfaces.

You must set the current design with `set_current_design` before using the `get_icl_scan_interface_list` command.

Arguments

- `-control_of object_spec`

An optional string that specifies an `object_spec` with at most one element, which is either an object of type `icl_scan_interface` or an object of type `icl_scan_register`.

The `-control_of` option cannot be used in the ICL extraction flow when there is no top level ICL.

Return Values

When this command is used without the `-control_of` option, then if the ICL top module exists, the command returns *the ScanInterface names* in the ICL top module. If the module does not have an explicit `ScanInterface` definition in the ICL module, but only an implicitly defined anonymous scan interface, the command returns “unnamed”. If the ICL top module does not exist, the command returns the *new ScanInterface names* specified with the `add_icl_scan_interfaces` command, which will be created during ICL extraction.

When the `-control_of` option is specified, the command returns a list of names (not a collection) of top level client scan interfaces, through which the given scan interface / the given scan register can be accessed.

Examples

The following example returns the ScanInterface names PG1 and PG2 in the ICL module pass1:

```
Module pass1 {
    ScanInPort sin1;
    ScanInPort sin2;
    ScanOutPort sout;
    ShiftEnPort sen;
    CaptureEnPort cen;
    SelectPort select1;
    SelectPort select2;
    UpdateEnPort uen;
    ScanInterface PG1 {
        Port sin1; Port sout; Port sen; Port cen; Port select1; Port uen;
    }
    ScanInterface PG2 {
        Port sin2; Port sout; Port sen; Port cen; Port select2; Port uen;
    }
}

set_current_design pass1
get_icl_scan_interface_list

PG1 PG2
```

Related Topics

[add_icl_scan_interfaces](#)
[delete_icl_scan_interfaces](#)
[set_icl_scan_interface_ports](#)

get_icl_scan_interface_port_list

Context: all contexts

Mode: all modes

Returns the *names of the ports for the specified* scan interface in an existing ICL top module, or the names of the ports *for the specified* scan interface that will be created in the new ICL top module during ICL extraction.

Usage

```
get_icl_scan_interface_port_list -name name
```

Description

Returns the *names of the ports for the specified* scan interface in an existing ICL top module, or the names of the ports *for the specified* scan interface that will be created in the new ICL top module during ICL extraction.

You must set the current design with `set_current_design` before using the `get_icl_scan_interface_port_list` command.

Arguments

- **-name name**

A required switch and value pair that specifies the name of a scan interface, or a collection that contains exactly one element of type either “`icl_scan_interface`” or “`icl_scan_interface_of_module`” that specifies the scan interface, for which the list of ports will be returned. If the module does not have an explicit `ScanInterface` definition in the ICL module, but only an implicitly defined anonymous scan interface, then use the string “`unnamed`” for the scan interface name to get the ports of the anonymous scan interface.

Return Values

If the ICL top module exists, this command returns *the names of the ports for the specified ScanInterface*.

If the ICL top module does not exist, the command returns the names of the new ports for the `ScanInterface` specified with the `set_icl_scan_interface_ports` command, which will be created during ICL extraction.

Examples

Example 1

This example returns the list of ports for the ScanInterface named default in the ICL module pass1:

```
Module pass1{
    ScanInPort sin;
    ScanOutPort sout;
    SelectPort select;
    CaptureEnPort cen;
    ShiftEnPort shiftEn;
    UpdateEnPort updateEn;
    ScanInterface default {
        Port sin; Port sout; Port select; Port cen; Port shiftEn; Port
updateEn; DefaultLoadValue 3'b0;
    }
}
```

```
set_current_design pass1
get_icl_scan_interface_port_list -name default
sin sout select cen shiftEn updateEn
```

Example 2

This example returns the list of ports for the anonymous scan interface in the ICL module pass2:

```
Module pass2{
    ScanInPort sin;
    ScanOutPort sout;
    SelectPort select;
    CaptureEnPort cen;
    ShiftEnPort shiftEn;
    UpdateEnPort UpdateEn;
}
```

```
set_current_design pass2
get_icl_scan_interface_port_list -name unnamed
sin sout select cen shiftEn updateEn
```

Related Topics

[add_icl_scan_interfaces](#)
[delete_icl_scan_interfaces](#)
[get_icl_scan_interface_list](#)
[set_icl_scan_interface_ports](#)

get_icl_scope

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Introspects the current ICL scope.

Usage

```
get_icl_scope [-iCall | -iPrefix]
```

Description

Introspects the current ICL scope.

When executed without an option, this command returns the instance name prefix that will be appended to the beginning of any ICL object names used in the [iCall](#), [iWrite](#), and [iRead](#) commands. When executed with the -iCall option, this command ignores the current iPrefix specification and returns the ICL scope against which the current iProc is running. When executed with the -iPrefix option, this command returns the current value specified for the iPrefix command.

For more information on the ICL scope, see the [iCall](#) command reference page.

Arguments

- **-iCall**
An optional literal that specifies to ignore the current iPrefix specification and return the ICL scope against which the current iProc is running. This path is useful to obtain the exact ICL instance the iProc was executed against using the iCall command. You can use this value in an iNote as shown in the example below.
- **-iPrefix**
An optional literal that specifies to return the current value specified for the iPrefix command.

Return Values

An ICL instance path name.

Examples

In the following example, the iProc gets the ICL scope in which it was executed against with the iCall command to echo into the retargeted pattern using an iNote command:

```
iProc myProc {} {  
    iNote "Running on Instance [get_icl_scope -iCall]"  
}
```

Related Topics

[iCall](#)

[iPrefix](#)

get_iclock_list

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Returns a list of ICL port and pin names on which you have issued an iClock command.

Usage

```
get_iclock_list [-overridden] [-within_current_pattern_set]
```

Description

Returns a list of ICL port and pin names on which you have issued an iClock command.

When you use the -overridden option, this command lists the ICL port and pin names on which you have issued an iClockOverride command on. You can use this list to introspect the effective source and the effective frequency multiplication and division values of the ICL pin or port using the [get_iclock_option](#) [-overridden] command.

Arguments

- `-overridden`

Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

- `-within_current_pattern_set`

Restricts the result of the command to the iClock/iClockOverride specifications that have been done within the currently opened pattern set.

Return Values

A Tcl list of ICL port and pin names.

Examples

In the following example, a Tcl list of ICL port and pin names are obtained using the `get_iclock_list` command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
    puts "iClock pin or port: $iclock"
    puts " source = [get_iclock_option $iclock -source]"
    puts " freq_multiplier= [get_iclock_option $iclock -freq_multiplier]"
    puts " freq_divider = [get_iclock_option $iclock -freq_divider]"
}
```

Related Topics

[add_clocks](#)
[delete_clocks](#)
[get_iclock_option](#)
[iClock](#)
[iClockOverride](#)
[report_clocks](#)
[report_icl_modules](#)

get_iclock_option

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified `icl_port` or `pin_name`.

Usage

```
get_iclock_option icl_port_or_pin_name [-overridden]
  {-source | -source_type | -freq_multiplier | -freq_divider |
   -period [-in {as | fs | ps | ns | us | ms | s}] | -traversed_pins | -error_list }
```

Description

Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified `icl_port` or `pin_name`.

The specified `icl_port_or_pin_name` must have been the target of an iClock command when the `-overridden` option is not specified; it must have been the target of an iClockOverride command when the `-overridden` option is specified.

Arguments

- ***icl_port_or_pin_name***

A required string that specifies the name of an ICL port or pin object on which an iClock or iClockOverride command has been issued.

- **-overridden**

An optional switch that specifies that `icl_port` or `pin_name` must be an `icl_port` or `icl_pin` object on which an iClockOverride command has been issued. If this switch is used, the return value of `get_iclock_option` is derived from the most recent matching iClockOverride command, instead of the most recent matching iClock command.

- **-source**

A required switch that specifies to return the effective source of an iClock ICL pin or port, or the specified source of an iClockOverride ICL pin or port.

- **-source_type**

A required switch that specifies to return the type of the effective source of an iClock ICL pin or port, or the specified source of an iClockOverride ICL pin or port. The result is one of the following: “primary”, “tck”, “internal” or “none”. The following table shows the meaning of the different source types:

Table 4-6. Source Types

type	without -override	with -override
primary	The effective source of the iClock pin or port is a ClockPort of the current design.	The specified source of the iClockOverride pin or port is a ClockPort of the current design.
tck	The effective source of the iClock pin or port is a TCKPort of the current design.	The specified source of the iClockOverride pin or port is a TCKPort of the current design.
internal	The effective source of the iClock pin or port is a ToClockPort with a “Period” property.	The iClockOverride command for the pin or port has a “-period” specification instead of a “-source” specification, or the “-source” specification refers to an internal ICL pin.
none	The specified ICL pin or port has not been subject to a successfully completed iClock command.	The specified ICL pin or port has not been subject to a successfully completed iClockOverride command.

The `-source_type` switch is special, in the sense that it can be used without error even if the specified ICL pin or port has not been subject to an iClock or iClockOverride command before. The return value “none” denotes this situation.

- **-freq_multiplier**

A required switch that specifies to return the effective frequency multiplier value of an iClock ICL pin or port, or the specified frequency multiplier value of an iClockOverride ICL pin or port.

- **-freq_divider**

A required switch that specifies to return the effective frequency divider value of an iClock ICL pin or port, or the specified frequency divider value of an iClockOverride ICL pin or port.

- **-period [-in {as | fs | ps | ns | us | ms | s}]**

A required switch that specifies to return the effective clock period of an iClock ICL pin or port, or the specified clock period of an iClockOverride ICL pin or port. You can optionally

use the -in switch and keyword to specify a particular time unit. For iClock specifications, the default time unit is derived from the time unit of the source clock. This is either the time unit used in the -period switch of the add_clocks command (in case of an asynchronous source clock) or the time unit of the tester period (in case of a synchronous source clock). For iClockOverride specifications, the default time unit is derived from the unit that has been used during the processing of the iClockOverride command.

- **-traversed_pins**

A required switch that specifies to return a dictionary of pins that have been traversed during the processing of the iClock command. When you use the -traversed_pins switch, you cannot also use the -overridden switch. The structure of the result is a dictionary of dictionaries as follows:

```

{   <pin_name_1> { node_type      port|pin
                    inversion     0|1
                    freq_multiplier <int>
                    freq_divider    <int>
}
<pin_name_2> { node_type      port|pin
                    inversion     0|1
                    freq_multiplier <int>
                    freq_divider    <int>
}
...
<pin_name_n> { node_type      port|pin
                    inversion     0|1
                    freq_multiplier <int>
                    freq_divider    <int>
}
}

```

The order of pins is from source to sink (iClock pin or port). Frequency multipliers, frequency dividers and inversions are accumulated appropriately.

- **-error_list**

A required switch that specifies to return a list of error messages associated with the iClock ICL pin or port. If the -overridden switch is used, the result is empty, because iClockOverride error messages are currently not collected.

Return Values

The value of the specified option for the specified icl_port_or_pin_name.

Examples

In the following example, a Tcl list of ICL port and pin names are obtained using the get_iclock_list command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
    puts "iClock pin or port: $iclock"
    puts " source = [get_iclock_option $iclock -source]"
    puts " freq_multiplier= [get_iclock_option $iclock -freq_multiplier]"
    puts " freq_divider = [get_iclock_option $iclock -freq_divider]"
}
```

Related Topics

[add_clocks](#)
[delete_clocks](#)
[get_iclock_list](#)
[iClock](#)
[iClockOverride](#)
[report_clocks](#)
[report_icl_modules](#)

get_ijtag_instances

Context: dft, patterns

Mode: all modes

Returns a collection of instances for which there is a matched ICL module.

Usage

```
get_ijtag_instances [ijtag_instance_spec] [-below_instances instance_objects]  
[-filter attribute_equation] [-silent]
```

Description

Returns a collection of instances for which there is a matched ICL module.

You can restrict the search to instances below a given collection of instances and filter the collection based on an arbitrary filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if no Ijtag instances are returned.

Arguments

- *ijtag_instance_spec*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all Ijtag instances.

If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- -below_instances *instance_objects*

An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The specified *name_patterns* are searched relative to each instance found in the *instance_objects* list.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the [-below_instances switch](#) of the [get_instances](#) command description for the justification of this behavior.

- -filter *attribute_equation*

An optional switch and string that specify to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the instance object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering using the attribute equation format.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This **-silent** option has no effect on any other error reporting.

Note

 You can use the **-silent** switch to manage the case in which an empty collection is returned. Before using the **-silent** switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

The following example uses the **get_ijtag_instances** command to get all instances of a module below a given instance for which there is a matched ICL file.

```
get_ijtag_instances -below_instance subblockA_i1
{subblockA_i1/tdr1 subblockA_i1/tdr2}
```

Related Topics

[get_ijtag_instance_option](#)
[set_ijtag_instance_options](#)
[report_ijtag_instances](#)

get_ijtag_instance_option

Context: dft, patterns

Mode: all modes

Returns the value of the selected option for the specified Ijtag instance.

Usage

```
get_ijtag_instance_option ijtag_instance_spec [-use_in_dft_specification]  
[-connect_data_signals_already_connected_to_ports]
```

Description

Returns the value of the selected option for the specified Ijtag instance.

Arguments

- *ijtag_instance_spec*

A required literal that is the name of an ijtag instance or a collection containing one ijtag instance as reported by [get_instances](#) or [get_ijtag_instances](#).

The instance object must have an ICL module matched to it.

- -use_in_dft_specification

A Boolean option that specifies that the value of the -use_in_dft_specification option is required. This value is set using the [set_ijtag_instance_options](#) and is used by the [create_dft_specification](#) command.

- -connect_data_signals_already_connected_to_ports

A Boolean option that specifies that the value of the -connect_data_signals_already_connected_to_ports option is required. This value is set using the [set_ijtag_instance_options](#) and is used by the [create_dft_specification](#) command.

Examples

The following example returns the value of the use_in_dft_specification option of all Ijtag instances.

```
foreach_in_collection inst [get_ijtag_instances] {  
    set opt [get_ijtag_instance_option $inst -use_in_dft_spec]  
    puts "inst = [get_single_name $inst], use_in_dft_spec = $opt"  
}  
  
inst = u1/trd1, use_in_dft_spec = auto  
inst = u1/trd2, use_in_dft_spec = auto  
inst = u2/trd1, use_in_dft_spec = off
```

Related Topics

[get_ijtag_instances](#)
[set_ijtag_instance_options](#)

[**report_ijtag_instances**](#)

get_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not issued.

Usage

```
get_ijtag_retargeting_options -test_setup_network_end_state  
    |-test_setup_ireset  
    |-apply_target_annotations  
    |-max_operations_per_iapply  
    |-compare_constant_capture_values  
    |-compare_constant_data_out_ports  
    |-merge_irunloop_only  
    |-inject_cycles  
    |-tck_ratio  
    |-annotation_parameter_values
```

Description

Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not issued.

To use this command, you must have previously loaded and elaborated the top-level ICL module. Otherwise, the tool issues an error message.

Arguments

- **-test_setup_network_end_state**

A switch that returns the value of the -test_setup_network_end_state option (keep or reset) specified by the previous set_ijtag_retargeting_options command or returns the default value (reset) if the set_ijtag_retargeting_options command was not issued.

- **-test_setup_ireset**

A switch that returns the value of the -test_setup_ireset option (off, first_icall or when_icl_present) specified by the previous set_ijtag_retargeting_options command or returns the default value (when_icl_present) if the set_ijtag_retargeting_options command was not issued.

- **iapply_target_annotations**

A switch that returns the value of the -iapply_target_annotations option (off, dense, or full) specified by the previous set_ijtag_retargeting_options command or returns the default value (dense) if the set_ijtag_retargeting_options command was not issued.

- **-max_operations_per_iapply**

A switch that returns the value of the `-max_operations_per_iapply` option specified by the previous `set_ijtag_retargeting_options` command or returns the default value (64) if the `set_ijtag_retargeting_options` command was not issued.

- **-compare_constant_capture_values**

A switch that returns the value of the `-compare_constant_capture_values` option (on or off) specified by the previous `set_ijtag_retargeting_options` command or returns the default value (on) if the `set_ijtag_retargeting_options` command was not issued.

- **-compare_constant_data_out_ports**

A switch that returns the value of the `-compare_constant_data_out_ports` option for the primary DataOut ports (on or off) specified by the previous `set_ijtag_retargeting_options` command or returns the default value (off) if the `set_ijtag_retargeting_options` command was not issued.

- **-merge_irunloop_only**

A switch that returns the value of the `-merge_irunloop_only` option (on or off) specified by the previous `set_ijtag_retargeting_options` command or returns the default value (off) if the `set_ijtag_retargeting_options` command was not issued.

- **-inject_cycles**

A switch that returns the value of the `-inject_cycles` option (Tcl list with an even number of arguments, each comprising a TAP state and the related number of injected cycles in alternating order) specified by the previous `set_ijtag_retargeting_options` command or returns the default value if the `set_ijtag_retargeting_options` command was not issued.

- **-tck_ratio**

A switch that returns the value of the `-tck_ratio` option (“auto” or a power of 2) specified by the previous `set_ijtag_retargeting_options` command or returns the default value (auto) if the `set_ijtag_retargeting_options` command was not issued.

- **-annotation_parameter_values**

A switch that returns the value of the `-annotation_parameter_values` option (Tcl list with an even number of arguments, each comprising a key and a value in alternating order, describing the configuration of notes, comments and annotations in Tesson IJTAG) specified by the previous `set_ijtag_retargeting_options` command, or returns the default value if the `set_ijtag_retargeting_options` command was not issued.

Examples

The following example loads the ICL, turns off the reset of the ICL network after the last `test_setup` PDL command and then displays the value of the selected option:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl ..../data/chip.icl
SETUP> set_current_design
```

```
// Note: Top design is 'chip'.  
  
SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep  
SETUP> get_ijtag_retargeting_options -test_setup_network_end_state  
keep
```

Related Topics

[report_ijtag_retargeting_options](#)

[set_ijtag_retargeting_options](#)

[get_input_constraints](#)

Context: dft, patterns

Mode: setup, analysis

Returns the present constraint status of primary input ports or pseudo ports.

Usage

```
get_input_constraints obj_spec { -equivalent_ports | -equivalent_inverted_ports |  
-constraint_value | -hold | -no_z | -slow_pad }
```

Description

The `get_input_constraints` command displays the constraint status of primary input pins to which you have applied constraints using the [add_input_constraints](#) command.

Arguments

- *obj_spec*

A required value that specifies a collection of ports and pseudo ports.

- `-equivalent_ports`

An optional switch that specifies to return the ports/pseudo ports that are defined as the non-inverted equivalent of the specified port/pseudo port. The *obj_spec* collection must contain exactly one port or pseudo port object.

- `-equivalent_inverted_ports`

An optional switch that specifies to return the ports/pseudo ports that are defined as the inverted equivalent of the specified port/pseudo port. The *obj_spec* collection must contain exactly one port or pseudo port object.

- `-constraint_value`

An optional switch that specifies to return the constraint values specified for the given ports/pseudo ports. The returned constraint values can be any of the following:

C0 — Constant 0

C1 — Constant 1

CX — Constant X (unknown)

CZ — Constant Z (high-impedance)

CR0 — Constant that returns to 0

CR1 — Constant that returns to 1

CT0 — Constant TIE0

CT1 — Constant TIE1

CTZ — Constant TIEZ

The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with *obj_spec*. The returned Tcl list length equals the number of ports/pseudo ports specified with *obj_spec*. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- **-hold**

An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -hold was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with *obj_spec*. The returned Tcl list length equals the number of ports/pseudo ports specified with *obj_spec*. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- **-no_z**

An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -no_z was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with *obj_spec*. The returned Tcl list length equals the number of ports/pseudo ports specified with *obj_spec*. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- **-slow_pad**

An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -slow_pad was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with *obj_spec*. The returned Tcl list length equals the number of ports/pseudo ports specified with *obj_spec*. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

Examples

Example 1

The following example adds constraints with the add_input_constraints command, then retrieves them with the report_input_constraints and get_input_constraints commands.

```
SETUP> add_input_constraints -equivalent A[2] B[1] B[0] -invert A[0] A[1]
```

```
SETUP> report_input_constraints
```

Primary Input	Equivalent	C-Type	Hold	No-Z	Slow-Pad
A[2]		NONE	No	Undetermined	No
	~A[1]	NONE	No	Undetermined	No
	~A[0]	NONE	No	Undetermined	No
	B[0]	NONE	No	Undetermined	No
	B[1]	NONE	No	Undetermined	No

```
SETUP> get_input_constraints A[0] -equivalent_ports
```

```
{A[1]}
```

```
SETUP> get_input_constraints A[1] -equivalent_ports
{A[0]}

SETUP> get_input_constraints A[2] -equivalent_ports
{B[0] B[1]}

SETUP> get_input_constraints A[0] -equivalent_inverted_ports
{A[2] B[0] B[1]}

SETUP> get_input_constraints A[2] -equivalent_inverted_ports
{A[0] A[1]}

SETUP> get_input_constraints A[2] -hold
0

SETUP> add_input_constraints A[4] -C1
SETUP> get_input_constraints A[12:0] -constraint_value
{} {} {} {} {} {} {} {} {} C1 {} {} {} {}

SETUP> get_input_constraints A[2] -no_z
0

SETUP> add_input_constraints A[2] -slow_pad
SETUP> get_input_constraints A[2] -slow_pad
1

SETUP> get_input_constraints {A[0] A[1] A[2]} -slow_pad
0 0 1

SETUP> add_input_constraints B[1] -CT1 -no_z
SETUP> get_input_constraints B[1] -constraint_value
CT1

SETUP> get_input_constraints B[1] -no_z
1
```

Example 2

The following examples check if a constraint is defined:

```
set conValue [get_input_constraints xyz/n_123 -constraint_value]
if { [llength [lindex $conValue 0]] != 0 } {
    puts "Constraint is defined."
} else {
    puts "Constraint not defined"
}
```

```
set ports [get_ports]
set constraints [get_input_constraints $ports -constraint_value]
for {set i 0} { $i < [llength $constraints] } { incr i } {
    if { [llength [lindex $constraints $i]] != 0 } {
        puts "Constraint is defined for port [get_name_list [index_collection $ports $i]]"
    } else {
        puts "Constraint not defined for port [get_name_list [index_collection $ports $i]]"
    }
}
```

get_insertion_option

Context: unspecified, all contexts

Mode: all modes

Returns the value of any insertion option that is specifiable with the [set_insertion_options](#) command.

Usage

```
get_insertion_option {-auto_uniquify_edited_modules
    |-edited_module_prefix
    |-net_uniquification_suffix
    |-module_uniquification_suffix
    |-instance_uniquification_suffix
    |-allow_assigns
    |-inserted_object_comment_label
    |-treat_common_ancestor_as_unique}
    |-vhdl_port_pin_net_type
    |-open_input_pins
    |-remove_inactive_sections_in_unrolled_generate_loops
```

Description

Returns the value of any insertion option that is specifiable with the [set_insertion_options](#) command.

Only one option can be specified per `get_insertion_option` invocation. The return value is a string containing the current setting of the specified option. An error is generated if more than one option is specified, and the return value will be null.

Arguments

- `-auto_uniquify_edited_modules`

A switch that returns the current setting of `-auto_uniquify_edited_modules` option of the [set_insertion_options](#) command.

- `-edited_module_prefix`

A switch that returns the current setting of the `-edited_module_prefix` option of the [set_insertion_options](#) command.

- `-net_uniquification_suffix`

A switch that returns the current setting of the `-net_uniquification_suffix` option of the [set_insertion_options](#) command.

- `-module_uniquification_suffix`

A switch that returns the current setting of the `-module_uniquification_suffix` option of the [set_insertion_options](#) command.

- [-instance_uniquification_suffix](#)
A switch that returns the current setting of the -instance_uniquification_suffix option of the [set_insertion_options](#) command.
- [-allow_assigns](#)
A switch that returns the current setting of the -allow_assigns option of the [set_insertion_options](#) command.
- [-inserted_object_comment_label](#)
A switch that returns the current setting of the -inserted_object_comment_label option of the [set_insertion_options](#) command.
- [-treat_common_ancestor_as_unique](#)
A switch that returns the current setting of the -treat_common_ancestor_as_unique option of the [set_insertion_options](#) command.
- [-vhdl_port_pin_net_type](#)
A switch that returns the current setting of the -vhdl_port_pin_net_type of the [set_insertion_options](#) command.
- [-open_input_pins](#)
A switch that returns the current setting of the -open_input_pins option of the [set_insertion_options](#) command.
- [-remove_inactive_sections_in_unrolled_generate_loops](#)
A switch that returns the current setting of the -remove_inactive_sections_in_unrolled_generate_loops option of the [set_insertion_options](#) command.

Examples

The following example introspects the current setting of the -edited_module_prefix option of the [set_insertion_options](#) command.

```
get_insertion_option -edited_module_prefix
```

```
MyCore_
```

Related Topics

[set_insertion_options](#)

get_insert_test_logic_option

Context: dft -scan, dft -test_points

Mode: setup, analysis, insertion

Retrieves a prefix/infix value that will be used when logic is created by the insert_test_logic command.

Usage

```
get_insert_test_logic_option
  [ -inserted_object_prefix ] |
  [ -persistent_cell_prefix ] |
  [ -persistent_clock_cell_prefix ] |
  [ -logic_type
    [ generic
    | clock_gater_fix
    | clock_control_fix
    | dedicated_input_wrapper
    | dedicated_input_holding_wrapper
    | dedicated_output_wrapper
    | lockup
    | pipeline
    | ram_fix
    | testpoint_control
    | testpoint_observation
    | tri_state_fix
    | x_bounding ] ]
```

Description

The get_insert_test_logic_option command retrieves a prefix/infix value that will be used when logic is created by the insert_test_logic command.

Arguments

- **-inserted_object_prefix**
An optional switch that returns the default prefix of HDL objects created during “insert_test_logic”.
- **-persistent_cell_prefix**
An optional switch that returns the default prefix of persistent cell instances created during “insert_test_logic”.
- **-persistent_clock_cell_prefix**
An optional switch that returns the default prefix of persistent clock cell instances created during “insert_test_logic”.

- **-logic_type**

An optional switch that returns the default infix of HDL objects created during “insert_test_logic” for a specified logic type. The possible logic types are as follows:

generic — Used for generic logic.
 clock_gater_fix — Used for clock gater control logic.
 clock_control_fix — Used for clock control logic.
 dedicated_input_wrapper — Used for dedicated input wrapper logic.
 dedicated_input_holding_wrapper — Used for dedicated input holding wrapper logic.
 dedicated_output_wrapper — Used for dedicated output wrapper logic.
 lockup — Used for lockup logic.
 pipeline — Used for pipeline logic.
 ram_fix — Used for ram control logic.
 testpoint_control — Used for control testpoint logic.
 testpoint_observation — Used for observation testpoint logic.
 tri_state_fix — Used for tristate control logic.
 x_bounding — Used for X-bounding logic

Examples

The following example demonstrates the use of the get_insert_test_logic_option command.

```
SETUP> get_insert_test_logic_option -logic_type dedicated_input_wrapper
diw_
SETUP> set_insert_test_logic_options -logic_type dedicated_input_wrapper \
        -logic_infix inreg
SETUP> get_insert_test_logic_option -logic_type dedicated_input_wrapper
inreg
SETUP> report_insert_test_logic_options
```

// 'insert_test_logic' Command Option	Current Value
// -----	-----
// -inserted_object_prefix	ts_
// -logic_type clock_control_fix	clkfix_
// -logic_type clock_gater_fix	cfgix_
// -logic_type dedicated_input_holding_wrapper	dihw_
// -logic_type dedicated_input_wrapper	inreg_
// -logic_type dedicated_output_wrapper	dow_
// -logic_type generic	logic_
// -logic_type lockup	lockup_
// -logic_type pipeline	pipeline_
// -logic_type ram_fix	ramfix_
// -logic_type testpoint_control	cp_
// -logic_type testpoint_observation	op_
// -logic_type tri_state_fix	trifix_
// -logic_type x_bounding	xb_
// -persistent_cell_prefix	tessent_persistent_cell_
// -persistent_clock_cell_prefix	tessent_persistent_clock_cell_

Related Topics

[set_insert_test_logic_options](#)

[report_insert_test_logic_options](#)

get_instances

Context: all contexts

Mode: all modes

Returns a collection of all instances instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

Usage

```
get_instances [name_patterns]
  [-of_modules module_objects]
  [-parent_of_instances instance_objects
    | -below_instances instance_objects
    | -of_pins pin_objects
    | -of_nets net_objects
    | -of_gate_pins gate_pin_objects
    | -of_icl_instances icl_instance_objects ]
  [-of_type types]
  [-hierarchical]
  [-match_rtl_reg | -use_path_matching_options]
  [-filter attribute_equation]
  [-regexp]
  [-nocase]
  [-silent]
```

Description

Returns a collection of all instances instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

When the *name_patterns* list is omitted, all instances are returned subject to filtering imposed by the specified options.

By default, the tool reports an error if the `get_instances` command returns no result unless you have set the `-silent` switch.

You can change no result Tcl errors to warnings by issuing the following command:

```
set_tcl_shell_options -change_no_result_warnings_to_errors off
```

Note

 This approach introduces a risk if your scripts cannot deal with empty collections correctly. As this may result in incorrect/unintended behavior, it is not recommended.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all instances.

If you specify one of the optional switches (-parent_of_instances, -of_pins, -of_nets, -of_gate_pins or -hierarchical), the name patterns are matched against the *leaf_name* attribute value of the instances in which case no hierarchy separators are allowed in the pattern list.

If you specify the -below_instances option, the name patterns are hierarchical name patterns relative to the specified instance objects otherwise the name patterns are relative to the current design.

- *-of_modules module_objects*

An optional switch and value pair that constrains the command to return only instances of modules specified by *module_objects*. The *module_objects* is a single module name, a Tcl list of module names, or a collection of module objects as returned by the [get_modules](#) command.

- *-parent_of_instances instance_objects*

An optional switch and value pair that constrains the command to return a collection of instance objects that are the parent instances of the specified *instance_objects*. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional *name_patterns* option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -parent_of_instances option is used.

- *-below_instances instance_objects*

An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The tool searches the specified *name_patterns* relative to each instance found in the *instance_objects* list. If the *instance_objects* is a null collection or a null string, the switch is ignored and instances are searched relative to the current design.

An exception is made for this switch to treat an empty collection as meaning the current design such that it could accept the return value of “`get_instances -parent_of_instance XXX`” even if XXX is directly instantiated into the current design. Very often, you want to create an instance or find objects in the same parent instance of another instance. If the -below_instances switch did not accept a null collection as meaning the current design, you could not use this simple form:

```
get_instances -below_instance [get_instances -parent_of_instance XXX]
```

You would always have to make an exception for instances found directly below the current design as follows:

```
set parent_instance [get_instances -parent_of_instance XXX]
if {[sizeof_collection $parent_instance] > 0} {
    get_instances
} else {
    get_instances -below_instances $parent_instance
}
```

Because an empty collection is interpreted as meaning the current design when passed to the -below_instances switch, you do not need the above exception code handling to handle instances directly instantiated in the current design differently from those instantiated at least two levels below the current design.

- **-of_pins *pin_objects***

An optional switch and value pair that constrains the command to return the instance on which the pins belong to. The returned collection is filtered to only return a given instance once even if it belongs to many pins in the *pin_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_pins option is used.

- **-of_nets *net_objects***

An optional switch and value pair that constrains the command to return the instance in which the net objects are located. The returned collection is filtered to only return a given instance once even if it is the parent instance of many nets in the *net_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_nets option is used.

- **-of_gate_pins *gate_pin_objects***

An optional switch and value pair that constrains the command to return the collection of instances to which the gates associated with the gate pins belong. The returned collection is filtered to only return a given instance once even if it belongs to many gate pins in the *gate_pin_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_gate_pins option is used.

If the specified *gate_pin* exists under one or many levels of `celldesign module instances, then the highest-level `celldesign module instance is returned. In addition, *gate_pins* directly under the root module return a “”as instance.

- **-of_icl_instances *icl_instance_objects***

An option switch and value pair that constrains the command to return the instances corresponding to the specified *icl_instance_objects*. The *icl_instances_objects* can be retrieved using the [get_icl_instances](#) command. If the tool is successful, it will return a

collection of these instances. If not, it will report an error unless the `-silent` switch is specified.

- `-of_type types`

An optional switch and value pair that constrains the tool to return only instances whose type attribute is one of the following: design, cell, or primitive. The types value is either a single type string or a Tcl list of type strings such as {design cell}.

- `-hierarchical`

An optional string that specifies to perform pattern matching relative to all instances found below the current design or below the specified *instance_objects* when the `-below_of_instances` option is specified. When this option is specified, the *name_patterns* argument cannot include slashes which are hierarchy separator symbols.

- `-match_rtl_reg`

An optional switch that performs register name-mapping using the rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option allows you to find the register with the register mapping rules. See [add_rtl_to_gates_mapping](#) for complete information.

- `-use_path_matching_options`

An optional switch that performs instance name-mapping using rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option allows you to find an instance using the instance mapping rules. See [add_rtl_to_gates_mapping](#) for complete information.

- `-filter attribute_equation`

An optional switch and string that specify to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the instance object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format.

- `-regexp`

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. You must properly escape the bracket “[]” symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- `-nocase`

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the *name_patterns*.

- `-silent`

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the

name_patterns disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

The following example returns a collection of all instances of cell types that start with ABC found anywhere below the current design.

```
get_instances ABC* -hierarchical -of_type cell
{u2/ABCD u1/u5/ABCDEF}
```

Example 2

The following example starts with the collection of all instances and filters it based on an attribute_equation of user-defined attributes.

```
get_instances -filter {myType1 || myType2 == "ABC"}
{u2 u1/u5}
```

Example 3

The following example starts with the collection of all instances and filters it based on the expression specified by the attribute_equation string.

```
get_instances -filter {name=~"u4*/u1*"
{u4/u1 u4/u5/u1}
```

Example 4

In the following example, the collection of all instances matching a list of hierarchical name patterns are additionally filtered to only return instances of type design. The names u1/* and u*/u* are searched relative to the current design.

```
get_instances {u1/* u*/u*} -of_type design
{u4/u5}
```

Example 5

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

```
get_instances u3* -hierarchical
{u3 u4/u3 u4/u5/u3 u5/u3}
```

Example 6

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* is matched against instances found below instance u4.

```
get_instances u3* -hierarchical -below_instances u4
```

```
{u4/u3 u4/u5/u3}
```

Example 7

In the following example, the -parent_of_instances option is used to find the parent instance of each instance on which a collection of pins exist. Only the parent instances whose leaf instance names match ABC* are returned. As described in the description of the -parent_of_instances switch, the optional name_pattern argument, in this case ABC*, is a string that is matched against the leaf instance name of every parent instance of the collection when the -parent_of_instances option is used.

```
set pin [get_pins]
get_instances ABC* -parent_of_instances [get_instances -of_pins $pin]
{u4/ABC1 u4/u5/ABC2}
```

Alternatively, you could use the base_name attribute of the pin objects and the parent_instance attribute of the instances to achieve the same result. However, the method used in this example is much more efficient because it deals with objects instead of strings containing object names. This method also avoids issues with string concatenation in the presence of escaped identifiers.

Example 8

This example returns all design instances that correspond to all ICL instances. It uses the [get_icl_instances](#) command to retrieve the ICL instances.

```
get_instances -of_icl_instances [get_icl_instances]
// { block1_I1/raw1_I1 block1_I1/raw1_I2 block1_I1/sib1_I1 block1_I1/sib1_I2 ...}
```

Related Topics

[add_rtl_to_gates_mapping](#)
[create_instance](#)
[delete_instances](#)
[get_common_parent_instance](#)
[rename_instance](#)
[replace_instances](#)
[report_rtl_to_gates_mapping](#)
[uniquify_instances](#)

get_instrument_dictionary

Context: all contexts including the unspecified one

Mode: all modes

Returns the Tcl dictionary created by the instrument processed by process_dft_specification or process_patterns_specification.

Usage

```
get_instrument_dictionary [dictionary_name [{key_name}...]] [-list] [-silent | -exists]
```

Description

Returns the content of a Tcl dictionary entry or the list of available dictionaries.

Use the -list option to see the currently available dictionary. At startup, you will only see the *tshell_global* dictionary. Once you have run the [process_dft_specification](#) command, other dictionaries will become available based on the instrument types that were processed.

The content of the dictionary is useful when you are writing customization scripts as described in the “[process_dft_specification.post_insertion Procedure](#)” on page 1176 section. For example, to know the extension of the RTL files that were created and stored in the [instruments](#) directory, use:

```
get_instrument_dictionary DftSpecification rtl_extension
```

For Tessent MissionMode in-system test, this command returns a dictionary named “mentor::in_system_test::PatternsSpecification” when you run the process_patterns_specification command to generate in-system test patterns. You can use the dictionary to help you program the CPU as described in “[Guidelines for Programming the CPU for In-System Test](#)” in the *Tessent MissionMode User’s Manual*.

See the “[Instrument Dictionary](#)” section for the description of the dictionary created by the boundary scan instrument.

Arguments

- *dictionary_name* [{key_name}...]

An optional dictionary name followed by any number of keys from that dictionary. If no keys are specified, then the entire dictionary is returned.

- *-list*

An optional Boolean switch that request the command to return the list of available instrument dictionaries. At startup, you will only see the *tshell_global* dictionary. Once you have run the [process_dft_specification](#) command, other dictionaries will become available based on the instrument types that were processed.

- **-silent**

An optional Boolean switch that suppresses the error that is normally produced when a dictionary or key is requested and the dictionary or key does not exist. With this switch, no errors are generated and the command returns a null value.

- **-exists**

An optional Boolean switch that request the command to return 1 if the specified dictionary and key combination exist and returns 0 otherwise.

Examples

Suppose you want to know the extension of the interface module created in the `dft_inserted_designs` directory when the current design is a Verilog module read in with format sv2005. You can query the dictionary using this command:

```
get_instrument_dictionary tshell_global Format2ExtensionMapping verilog sv2005
```

The command returns “sv05” and the interface module has the extension `sv05_interface`. To see the entire mapping list, use the following command:

```
format_dictionary [get_instrument_dictionary tshell_global Format2ExtensionMapping]
```

which returns the complete sub-dictionary.

The following example uses the -list switch to see the available dictionaries. It then shows how to display the entire “tshell_global” dictionary and its format using the `format_dictionary` command. The example then returns the value of the “Format2ExtensionMapping verilog 1995” key. The effect of the -silent and -exists switches are them shown.

```
get_instrument_dictionary -list
```

```
tshell_global
```

```
format_dictionary [get_instrument_dictionary tshell_global]
```

```
Extension2FormatMapping {
    verilog {
        v95 1995
        v 2001
        sv31a sv31a
        sv05 sv2005
        sv09 sv2009
    }
    vhdl {
        vhd87 1987
        vhd 1993
        vhd02 2002
        vhd08 2008
    }
}
Format2ExtensionMapping {
    verilog {
        1995 v95
        2001 v
        sv31a sv31a
        sv2005 sv05
        sv2009 sv09
    }
    vhdl {
        1987 vhd87
        1993 vhd
        2002 vhd02
        2008 vhd08
    }
}
Extension2DesignFormatMapping {
    v95 verilog_1995
    v verilog_2001
    sv31a verilog_sv31a
    sv05 verilog_sv2005
    sv09 verilog_sv2009
    vhd87 vhdl_1987
    vhd vhdl_1993
    vhd02 vhdl_2002
    vhd08 vhdl_2008
}
allowed_design_formats {verilog_1995 verilog_2001 verilog_sv31a
verilog_sv2005 verilog_sv2009 vhdl_1987 vhdl_1993 vhdl_2002 vhdl_2008}
date_time {
    format_string {
        %a %b
        %d %H:%M:%S
        %Z %Y
    }
    timezone {
    }
}
```

get_instrument_dictionary tshell_global Format2ExtensionMapping verilog 1995

v95

get_instrument_dictionary tshell_global junk

```
// Error: The specified dictionary 'tshell_global' does not contain the
// requested key 'junk'.
```

```
get_instrument_dictionary tshell_global junk -silent
```

```
get_instrument_dictionary tshell_global junk -exists
```

```
0
```

Related Topics

[format_dictionary](#)

[process_dft_specification](#)

get_instrument_parent_icl_module_list

Context: dft and patterns

Mode: analysis, insertion, and setup

Returns the parent or grandparent module names for ICL instance module names that match the specified *instrument_module_name*.

Usage

```
get_instrument_parent_icl_module_list instrument_module_name
```

Description

Returns the parent or grandparent module names for ICL instance module names that match the specified *instrument_module_name*. This command is useful to find all the actual ICL module names when it has been unqualified.

The tool returns the grandparent module name only when the `icl_module` attribute of the `tessent_instrument_subtype` of the ICL instances is `shared_bus_assembly`.

```
foreach module_name [get_instrument_parent_icl_module_list <instrument_controller_name> {  
    iProcsForModule $module_name  
    iProc ...  
}
```

The tool automatically inserts the command in the generated PDL files of the instrument.

Arguments

- ***instrument_module_name***

A required string that specifies the module name to use in the module name matching.

get_iproc_argument_default

Context: all contexts

Mode: all modes

Returns the default value for the specified *arg_name* of the specified *proc_name* attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by *module_name*.

Usage

```
get_iproc_argument_default proc_name arg_name [-exists] [-of_module module_name]  
[-silent]
```

Description

Returns the default value for the specified *arg_name* of the specified *proc_name* attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by *module_name*.

This command returns a null string when the specified argument has no default value. If you specify the *-exists* option, this command returns true if the specified argument has a default value and returns false when it does not.

Arguments

- ***proc_name***
A required string that specifies the name of the iProc.
- ***arg_name***
A required string that specifies the name of the argument.
- ***-exists***
An optional switch that changes the returned value of the command. When present, the command returns true when the specified argument has a default value and returns false when it does not. When not specified, the command returns the default value when it exists; otherwise it returns null.
- ***-of_module module_name***
An optional switch and value pair that constrains the command to return only the default value for the specified *arg_name* of the specified *proc_name* attached to the module specified by *module_name*.
- ***-silent***
An optional switch that specifies to suppress error messages if the specified *proc_name*, *arg_name* or *module_name* do not exist.

Return Values

If the `-exist` option is specified, this command returns true if the specified argument has a default value and returns false when it does not. If the `-exist` option is not specified, the command returns the default value when it exists; otherwise it returns null.

Examples

The following example lists each argument attached to the P1 iProc on module mychip and lists its default value, if one exists; otherwise it reports the argument does not have a default value.

```
foreach arg [get_iproc_argument_list P1 -of_module mychip] {  
    if {[!get_iproc_argument_default P1 $arg -exist]} {  
        puts "Argument $arg has no default value"  
    } else {  
        set def [get_iproc_argument_default P1 $arg]  
        puts "Argument $arg has default value of $def"  
    }  
}
```

Related Topics

[delete_iprocs](#)
[get_iproc_argument_list](#)
[get_iproc_list](#)
[iProc](#)
[IProcsForModule](#)

[get_iproc_argument_list](#)

Context: all contexts

Mode: all modes

Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last [iProcsForModule](#) command, or to the ICL module specified by `module_name`.

Usage

```
get_iproc_argument_list iproc_name [-of_module module_name] [-silent]
```

Description

Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last [iProcsForModule](#) command, or to the ICL module specified by `module_name`.

Arguments

- ***iproc_name***
A required string that specifies the name of the iProc for which the arguments are returned.
- **-of_module *module_name***
An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by `module_name`.
- **-silent**
An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Values

A Tcl list that contains the argument names of the specified iProc.

Examples

The following example assigns the arguments attached to the chiptest iProc on module mychip to the variable `chiptest_args`:

```
set chiptest_args [get_iproc_argument_list chiptest -of_module mychip]
```

Related Topics

[delete_iprocs](#)

[get_iproc_list](#)

[iProc](#)

[iProcsForModule](#)

get_iproc_body

Context: all contexts

Mode: all modes

Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

```
get_iproc_body iproc_name [-of_module module_name] [-silent]
```

Description

Returns the body for the specified iProc attached to the ICL module that was specified by the last [iProcsForModule](#) command, or to the ICL module specified by module_name.

Arguments

- *iproc_name*
A required string that specifies the name of the iProc.
- -of_module *module_name*
An optional switch and value pair that constrains the command to return only the body of iProcs attached to the module specified by module_name.
- -silent
An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Values

None

Examples

The following example returns the body of the chiptest iProc attached to module mychip:

```
get_iproc_body chiptest -of_module mychip
```

Related Topics

[delete_iprocs](#)

[get_iproc_argument_list](#)

[get_iproc_list](#)

[iProc](#)

[iProcsForModule](#)

get_iproc_list

Context: all contexts

Mode: all modes

Returns a Tcl list of iProcs attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by `module_name`.

Usage

```
get_iproc_list [-of_module module_name] [-silent]
```

Description

Returns a Tcl list of iProcs attached to the ICL module specified by the last [iProcsForModule](#) command, or to the ICL module specified by `module_name`.

Arguments

- `-of_module module_name`
An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by `module_name`.
- `-silent`
An optional switch that specifies to suppress error messages if the specified module does not exist.

Return Values

A Tcl list of the iProc names, which are registered against the ICL module specified by the last [iProcsForModule](#) command or the ICL module specified by `module_name`.

Examples

The following example assigns the list of iProcs attached to the mychip ICL module to the variable `myprocs`:

```
set myprocs [get_iproc_list -of_module mychip]
puts $myprocs
```

Related Topics

[delete_iprocs](#)
[get_iproc_argument_list](#)
[iProc](#)
[iProcsForModule](#)

get_layout_core_instance

Context: patterns -scan_diagnosis

Mode: setup, analysis

Returns the current layout core instance that was set with the `set_layout_core_instance` command.

Usage

`get_layout_core_instance`

Description

Returns the current layout core instance that was set with the `set_layout_core_instance` command.

Arguments

None

Related Topics

[report_scan_polygons](#)

[set_layout_core_instance](#)

get_license_queue_timeout

Context: all contexts

Mode: all modes

Returns the value specified by the previous set_license_queue_timeout command.

Usage

```
get_license_queue_timeout
```

Description

Returns the value specified by the previous [set_license_queue_timeout](#) command. The command returns the default value of “unlimited” if you have not issued the set_license_queue_timeout command in the current tool session.

Arguments

None

Examples

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3  
SETUP> get_license_queue_timeout
```

3

Related Topics

[set_license_queue_timeout](#)

get_loadboard_loopback_option

Context: patterns -ijtag, -scan, -scan_diagnosis, -scan_retargeting

Modes: Setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Returns the settings of options specified with the add_loadboard_loopback_pairs command.

Usage

```
get_loadboard_loopback_option {-inputs | -outputs | -ac_delay_to_z}
```

Description

The get_loadboard_loopback_option returns the setting of a single option specified with the [add_loadboard_loopback_pairs](#) command

Arguments

- **-inputs**
An optional switch that returns a collection of all input ports that are part of a loopback.
- **-outputs**
An optional switch that returns a collection of all output ports that are part of a loopback.
- **-ac_delay_to_z**
An optional switch that returns the ac_delay.

Examples

This example shows how you use this command to get each option set in the add_loadboard_loopback_pair:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]} \
    -ac_delay_to_z 10ns
get_loadboard_loopback_option -inputs
{din[4] ue}
get_loadboard_loopback_option -outputs
{dout[5] dout[0]}
get_loadboard_loopback_option -ac_delay_to_z
10ns 10ns
```

[get_logfile](#)

Context: unspecified, all contexts

Mode: all modes

Returns the name of and pathway to the current log file.

Usage

`get_logfile`

Description

Returns the name of and pathway to the current logfile as set by the tool invocation's `-logfile` *logfile_name* switch and value, or the value you set using the [set_logfile_handling](#) command.

Arguments

None.

Related Topics

[set_logfile_handling](#)

[tesson](#)

get_logical_library_list

Context: all contexts

Mode: all modes

Prerequisites: You must be in the -rtl context.

Returns a Tcl list with all logical design library names defined with the [set_logical_design_libraries](#) command.

Usage

```
get_logical_library_list [-default_library]
```

Description

Returns a Tcl list with all logical design library names defined with the [set_logical_design_libraries](#) command.

This command and the concept of logical libraries is only available in the RTL mode.

This command is useful for writing scripts that operate on all libraries such as in Example3 of the [get_modules](#) command reference.

Arguments

- `-default_library`

An optional switch that specifies that the returned list only contains the default library.

Return Values

A Tcl list with all logical design library names defined with the [set_logical_library_list](#) command.

Examples

In the following example, the Tcl variable `libs` is defined by the result returned by the [get_logical_library_list](#) command. You can then use a `foreach` command to iterate on the list elements.

```
set libs [get_logical_library_list]
```

Related Topics

[set_logical_design_libraries](#)

get_memory_instances

Context: dft

Mode: all modes

Prerequisite: The current design must be set with the `set_current_design` command.

Returns a collection of the instances for which there is a matched TCD Memory description.

Usage

```
get_memory_instances [name_patterns] [-below_instances instance_objects]  
[-filter attribute_equation] [-silent]
```

Description

Returns a collection of instances for which there is a matched TCD Memory description. See “[Tessent Core Description](#)” on page 3755 for more information.

You can restrict the search to instances below a given collection of instances, and filter the collection based on an arbitrary filtering equation involving other attributes registered against or inherited by the instance object type.

If none of the options of this command is used, the command returns a collection that contains all of the memory instances in the design.

The `-silent` option suppresses the warning that is normally generated if no memory instances are returned.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces `{}`. If no *name_patterns* are specified, the tool searches all memory instances.

If you specify the `-below_instances` option, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise, the name patterns are relative to the current design.

- `-below_instances instance_objects`

An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The specified *name_patterns* are searched relative to each instance found in the *instance_objects* list.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the `-below_instances` switch of the [get_instances](#) command description for the justification of this behavior.

- **-filter attribute_equation**

An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering using the attribute equation format.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

The following example uses the get_memory_instances command to return all instances of a module below a given instance for which there is a TCD Memory description.

```
get_memory_instances -below_instance subblockA_i1
{subblockA_i1/mem1 subblockA_i1/mem2}
```

Related Topics

[get_memory_instance_option](#)
[set_memory_instance_options](#)
[report_ijtag_instances](#)

[get_memory_instance_option](#)

Context: all contexts

Mode: all modes

Prerequisite: The current design must be set with the [set_current_design](#) command.

Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.

Usage

```
get_memory_instance_option {memory_instance [-bist_data_in_pipeline |  
-physical_cluster_override | -power_domain_island | -test_clock_override |  
-use_in_memory_bist_dft_specification | -use_in_memory_bisr_dft_specification]} |  
-bisr_segment_order_file | -physical_cluster_size_ratio
```

Description

Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.

Arguments

- *memory_instance*

A required string that specifies the name of a memory instance or a collection containing a single memory instance. You can use the “[foreach_in_collection](#) inst [\[get_memory_instances\]](#)” command to iterate on all memory instances. This option can be used with any option other than -bisr_segment_order_file or -physical_cluster_size_ratio.

- -bist_data_in_pipeline

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options](#) -bist_data_in_pipeline” command on the specified memory_instance.

- -physical_cluster_override

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options](#) -physical_cluster_override” command on the specified memory_instance.

- -power_domain_island

A Boolean option that specifies to return the computed power domain island for the specified memory_instance. This property is only defined when a UPF or CPF file was initially read in using the [read_cpf](#) or [read_upf](#) commands and you are in the analysis system mode. Two memories share a common power domain island name when they have the same [power_domain_name](#) and the two instances are not logically separated by instances having a different power_domain_name. For example, memories in u1 and memories in u1/u2/u3 sharing a common power_domain_name will be in two power domain islands if u1/u2 is associated with a different power_domain_name.

- **-test_clock_override**

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options -test_clock_override](#)” command on the specified memory_instance.

- **-use_in_memory_bist_dft_specification**

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options -use_in_memory_bist_dft_specification](#)” command on the specified memory_instance. In analysis system mode, the value auto is replaced by on or off depending on whether the memory instance is already connected to a memory BIST interface module.

- **-use_in_memory_bisr_dft_specification**

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options -use_in_memory_bisr_dft_specification](#)” on the specified memory_instance. In analysis system mode, the value auto is replaced by on or off depending on whether the memory instance is already connected to a memory BISR register module.

- **-bisr_segment_order_file**

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options -bisr_segment_order_file](#)” command. This option cannot be used together with the memory_instance option because *it applies to all memory instances*.

- **-physical_cluster_size_ratio**

A Boolean option that specifies to return the value previously specified with the “[set_memory_instance_options -physical_cluster_size_ratio](#)” command. This option cannot be used together with the memory_instance option because *it applies to all memory instances*.

Examples

The following example perform a query using the -use_in_memory_bist_dft_specification option in both setup and analysis mode. Notice how the value auto is converted to on in the analysis mode. This means the memory instance is not already connected to a memory BIST interface module.

```
SETUP> get_memory_instance_option -use_in_memory_bist_dft_specification
```

```
auto
```

```
SETUP> check_design_rules
```

```
...
```

```
ANALYSIS> get_memory_instance_option -use_in_memory_bist_dft_spec
```

```
on
```

Related Topics

[set_memory_instance_options](#)

[set_dft_specification_requirements](#)
[report_ijtag_instances](#)

get_module_parameter_list

Context: all contexts

Mode: all modes

Returns the list of parameters in a module.

Usage

```
get_module_parameter_list module_object [-overridden]
```

Description

Returns the list of parameters in a module.

There are restrictions you need to consider when you use this command on a module without a definition

- Only parameters that are specified on instantiation can be determined.
- The returned list of parameters is the same whether or not you use the -overridden switch.
- When parameters are not specified by name, but by order, the tool will generate a name. The generated name is a sequential integer, starting at zero, indicating the position of the parameter on the instantiation.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the [set_context](#) command.

Arguments

- ***module_object***

A required string that specifies the parameterized module objects whose parameters are to be returned.

- -overridden

An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Values

List of parameters.

Examples

Example 1

In the following example, the [get_module_parameter_list](#) is used to obtain the list of parameters overridden in the specific instantiation.

```
set inst u1/myinst_i1
set params [get_module_parameter_list [get_modules -of_instances $inst] -overridden]
```

Example 2

This example shows the names generated by the tool when parameters are not specified by name but by order.

```
set mods [get_module -of_instance [get_instances]]
foreach_in_collection mod $mods {
    puts "--- Module [get_single_name $mod] ---"
    foreach param [get_module_parameter_list $mod] {
        puts "parameter $param = [get_module_parameter_value $param -of_module $mod]"
    }
}

--- Module DW01_add@PV1 ---
parameter 0 = 8
--- Module DW01_add@PV2 ---
parameter 0 = 16
--- Module def_named@PV1 ---
parameter d_size = 3
parameter q_size = 4
--- Module def_ordered@PV1 ---
parameter d_size = 2
parameter q_size = 5
--- Module nodef_named@PV1 ---
parameter d_size = 3
parameter q_size = 4
--- Module nodef_ordered@PV1 ---
parameter 0 = 2
parameter 1 = 5
```

get_module_parameter_value

Context: all contexts

Mode: all modes

Returns the value of the parameter on the specified module object.

Usage

```
get_module_parameter_value parameter_name [-of_module module_object]
```

Description

Returns the value of the parameter on the specified module object.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the [set_context](#) command.

Arguments

- *parameter_name*
A required string that specifies the name of the parameter whose value is to be returned.
- *-of_module module_object*
An optional switch and string pair that specifies the design module whose parameter value is to be returned. When omitted, the parameters of the current design are introspected.

Return Values

The value of the requested design module parameter.

Examples

In the following example, the values of all overridden parameters are echoed in the transcript.

```
set inst [get_instance u1/myinst_i1]
set mod [get_modules -of_instances $inst]
set params [get_module_parameter_list $mod -overridden]
foreach param $params {
    puts "parameter $param = [get_module_parameter_value $param \
        -of_mod $mod]"
}
```

get_modules

Context: all contexts

Mode: all modes

Returns a collection of all modules that match the specified *name_patterns* list.

Usage

```
get_modules [name_patterns]
  [-in_library library] [-vhdl_architecture architecture]
  [-of_instances instance_objects] | -below_instances instance_objects | -of_ports port_objects
  | -parameterized_views_of_modules module_objects | -use_module_matching_options
  | -original_module_of module]
  [-of_type type]
  [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all modules that match the specified *name_patterns* list.

By default, this command returns all modules below the current design, and all modules present in memory, for example those modules you have read into the tool but are not yet instantiated. You can use the *-of_instances* and *-below_instances* arguments to constrain the search. If no modules exist in memory, the command returns an empty collection.

You can have any number of wildcards or regular expressions in the name patterns.

In the rtl context, you may have more than one logical library. Because VHDL is also supported in the rtl context, you may also have multiple architectures. You define the list of available logical libraries using the [set_logical_design_libraries](#) command.

In the no_rtl context, only one library exists and is it named “work”.

The tool searches for the objects specified by the *name_patterns* argument in all libraries unless the *-in_library* option is specified. All other commands that have an option that refers to module names only search the default library. For example, you use the following command to create an instance of module M1 from library L1:

```
create_instance instance_name -of_module [get_module M1 -in_library L1]
```

In another example, an instance of module M1 from the default library is created because only the *get_modules* command knows how to look into other libraries.

```
create_instance instance_name -of_module M1
```

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to match the returned list of modules against. If no *name_patterns* are specified, the tool searches all modules.

- *-in_library library*

An optional switch and value pair that specifies to restrict the search to a specified library. The library value must be a single string corresponding to one of the logical library names defined with the [set_logical_design_libraries](#) command. When the *-in_library* option is not specified, all libraries are searched. This option is only allowed in the rtl context.

Using *-in_library XXX* is equivalent to using *-filter “library_name == XXX”*.

- *-vhdl_architecture architecture*

An optional switch and value pair that specifies to restrict the search to a specific VHDL architecture name. This option is only useful when searching VHDL modules. When this option is not specified, one module per available architecture is returned. This option is only allowed in the rtl context.

Using *-vhdl_architecture XXX* is equivalent to using *-filter “vhdl_architecture == XXX”*.

- *-of_instances instance_objects*

An optional switch and value pair that constrains the command to return only modules of the specified *instance_objects*. You could use the *module_name* attribute of the instance objects to perform a similar filtering but this method is more direct and efficient and automatically deals with libraries and architectures.

- *-below_instances instance_objects*

An optional switch and value pair that constrains the command to return only modules corresponding to instances found below the specified *instance_objects*.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the [-below_instances](#) switch of the [get_instances](#) command description for the justification of this behavior.

- *-of_ports port_objects*

An optional switch and value pair that constrains the command to return only modules of the specified *port_objects*. You could perform a similar filtering operation using the *module_name* attribute of the port objects but using this switch is more direct and efficient and automatically deals with libraries and architectures. Note, this switch cannot be used with the *-of_instances* or *-below_instances* switch.

- *-parameterized_views_of_modules module_objects*

An optional switch and value pair that constrains the command to return a collection of modules corresponding to all the parameterized views of the module objects specified. The *module_objects* can be specified as a Tcl list of module names or a collection of modules.

Whether *module_objects* contains a parameterized view or the master copy of the module, this switch instructs the command to find all parameterized views and return them. Using this switch is equivalent to using the *master_name* attribute of a module and requesting all modules having the same *master_name* value. Using this switch, however, is much more efficient and it also handles logical libraries. You use this switch to determine whether a module is uniquely instantiated or not and to find out if a module has multiple parameterized views.

- **-use_module_matching_options**

An optional switch that constrains the command to return a collection of modules by applying all combinations of the suffix and the prefix defined by the *set_module_matching_options* command. This option is useful when you are looking for modules by names which were potentially unqualified during synthesis. When you use this switch, the *name_patterns* list cannot be empty and cannot include wild cards and the *-regexp* and *-nocase* switches are ignored.

- **-original_module_of module**

An optional switch and value pair that returns the original module from which the module was copied. A module is copied from another module either by unification or with the [copy_module](#) command. The *module* parameter can be the name of a single module or a collection of one module. The module may be a parameterized view of the copied module. The tool returns the master copy of the original module. If the specified *module* is not a copied module, the tool issues an error unless the *-silent* switch is used in which case an empty collection is returned.

- **-of_type type**

An optional switch and value pair that constrains the command to return only modules of the specified types. The type is a Tcl list of one or more strings matching one of the following values: design or cell.

- **-filter attribute_equation**

An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the module object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The *=~* and *!~* matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the *-regexp* option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This **-silent** option has no effect on any other error reporting.

Note

 You can use the **-silent** switch to manage the case in which an empty collection is returned. Before using the **-silent** switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

The following example looks for modules whose name matches a* or b*.

```
get_modules {a* b*}
{ab abc b bcd}
```

Example 2

The following example looks for modules having one or two characters.

```
get_modules {[::alnum::]{1,2}} -regexp
{ab b}
```

Example 3

The following example gets the module object associated to an instance.

```
get_modules -of_instances u3/u5
{myModule}
```

Example 4

The following example looks for modules in all libraries and displays their names and language value in a tabular format.

```

set format_string " "
append format_string "module_name = %-10s "
append format_string "language = %-10s "

set libs [get_logical_library_list]foreach lib $libs {
    puts "Modules in library $lib are:"
    set mods [get_modules -in_library $lib]
    foreach_in_collection mod $mods {
        puts [format $format_string \
            [get_att_val $mod -name name]\
            [get_att_val $mod -name language]]
    }
}

Modules in library lib1 are:
    module_name = dut           language = verilog
Modules in library lib2 are:
    module_name = dut           language = verilog
Modules in library work are:
    module_name = top          language = verilog

```

Example 5

This example shows how to report the names of the parameterized views for a module.

```

puts “[get_name_list [get_modules -parameterized_views_of_modules ff]]”
ff ff@PV1 ff@PV2

```

Example 6

The `-use_module_matching_options` switch is used to look for modules based upon the prefix and suffix values defined by the [set_module_matching_options](#) command.

```

//Setting the prefix and suffix module matching option
set_module_matching_options -prefix_pattern_list {a*}\ -suffix_pattern_list {_1 _2}
// Getting the modules
puts [join [get_name_list [get_modules {mymodule}\ -use_module_matching_options]] "\n"]
//The returned modules are:
a_mymodule
a_mymodule_1
a_mymodule_2
ab_mymodule
ab_mymodule_1
ab_mymodule_2
mymodule
mymodule_1
mymodule_2

```

Related Topics

[create_module](#)

get_multiprocessing_option

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Returns the value of a single specified variable previously set with the [set_multiprocessing_options](#) command.

Usage

```
get_multiprocessing_option {-generic_delete | -generic_scheduler  
| -license_timeout  
| -lsf_heuristics | -lsf_learning | -lsf_options | -multithreading  
| -processors_per_grid_request | -remote_shell | -result_time_limit  
| -scheduler_timeout | -sge_options}
```

Description

Returns the value of a single specified variable previously set with the [set_multiprocessing_options](#) command.

The `get_multiprocessing_option` command returns the value of a single multiprocessing variable that affects the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the [set_multiprocessing_options](#) command. To display the values of multiprocessing variables in a human-readable format, use the [report_multiprocessing_options](#) command.

For more information about setting up for multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual* and “[The Tessent Tcl Interface](#)” in the *Tessent Shell User’s Manual*.

Arguments

The arguments for the `get_multiprocessing_option` command are identical to those of the [set_multiprocessing_options](#) command, described on [page 2267](#). For more information about the purpose of each switch, refer to the list of arguments there.

Examples

The following example sets and then returns the value of the `remote_shell` variable:

```
set_multiprocessing_options -remote_shell rsh  
puts [get_multiprocessing_option -remote_shell]  
rsh
```

Related Topics

[add_processors](#)

[delete_processors](#)

[report_multiprocessing_options](#)
[report_processors](#)
[set_multiprocessing_options](#)

get_name_list

Context: unspecified, all contexts

Mode: all modes

Returns the name(s) of the specified object(s).

Usage

```
get_name_list object_spec [-remove_escaping] [-silent]
```

Description

Returns the name(s) of the specified object(s).

This command returns a list with the value of the name attribute for the objects specified by *object_spec*. Using this command is equivalent to executing the following command:

```
get_attribute_value_list object_spec -name name
```

The -remove_escaping argument removes the \ and the trailing spaces in an escaped identifier. Many legacy commands such as add_display_instance require the instance name to have the escape character stripped from its name. However, these legacy commands do accept collections as inputs in cases where the name format is not important. See examples below for an illustration of this behavior.

If an object in *object_spec* does not exist, an error message displays unless the -silent switch is specified.

Arguments

- *object_spec*

A required string that specifies a Tcl list of one or more object names, or a collection of one or more objects.

- -remove_escaping

An optional switch that specifies to strip out the \ and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See Example 2 for an illustration of this usage.

- -silent

An optional switch that specifies to suppress error messages if an object in *object_spec* does not exist.

Examples

Example 1

The following example retrieves the names for a collection of pins.

```
get_name_list [get_pins u1/i*]
```

```
{u1/i1 u1/i2 u1/i3}
```

Example 2

The following example illustrates the use of the `-remove_escaping` option. In this example, you can see how legacy commands natively support collections as input arguments but do not support names with escaped identifiers.

Notice the use of the double set of braces when specifying a name pattern containing escape identifiers to `get_pins`. These braces are needed to prevent the Tcl shell from interpreting the `\` character as its own escape sequence. You could also escape the `\` using `{u2\\u1/u2 /p*}` but using two sets of braces is easier when the pattern includes many slashes. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a complete description of pattern syntax and suggestions to use when the pattern is targeted for names containing escape characters.

```
get_pins {{u2\u1/u2 /p*}}
{{u2/\u1/u2 /p1} {u2/\u1/u2 /p2}}

add_display_instance [get_pins {{u2\u1/u2 /p*}}] -display hierarchical_schematic
set names [get_name_list [get_pins {{u2\u1/u2 /p*}}]]
{{u2/\u1/u2 /p1} {u2/\u1/u2 /p2} }

add_display_instance $names -display hierarchical_schematic
// Error: Specified instance does not exist or is not available at this
// level of hierarchy.
// Error: Incorrect argument /u2/\u1/u2 /p1.
// Error: Specified instance does not exist or is not available at this
// level of hierarchy.
// Error: Incorrect argument /u2/\u1/u2 /p2.

set names [get_name_list [get_pins {{u2\u1/u2 /p*}}] -remove_escaping]
{{u2/u1/u2/p1} {u2/u1/u2/p2} }

add_display_instance $names -remove_escaping -display hierarchical_schematic
```

Related Topics

- [get_attribute_list](#)
- [get_attribute_option](#)
- [get_attribute_value_list](#)
- [get_common_parent_instance](#)
- [get_modules](#)
- [get_icl_ports](#)

get_nets

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

Usage

```
get_nets [name_patterns] [-below_instances instance_objects]
           [-of_pins pin_objects | -of_ports port_objects | -of_gate_pins gate_pin_objects
           | -of_pseudo_ports pseudo_port_objects]
           [-hierarchical] [-filter attribute_equation] [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

When the *name_patterns* list is omitted, all nets are returned subject to filtering imposed by the specified options.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of nets. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all nets. In Usage 2 described previously, the name patterns are not allowed to include slashes which are hierarchy separator symbols. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a complete description of the pattern syntax when the -regexp option is used and not used.

- -below_instances *instance_objects*

An optional switch and value pair that constrains the command to search for nets below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the [-below_instances switch](#) of the [get_instances](#) command description for the justification of this behavior.

- -of_pins *pin_objects*

An optional switch and value pair that constrains the command to return the net objects associated to the *pin_objects*. Those are the nets having the exact same name as the pins and residing inside the instances on which the pins are on. For pins of cells, the inside nets corresponding to the pins are not visible and thus not returned. You cannot use a

name_patterns list when using this option. To get the net connected to pins from the outside, use `get_fanin` and `get_fanout -stop_on net`.

- **-of_ports port_objects**

An optional switch and value pair that constrains the command to return the net objects associated to the `port_objects`. Those are the nets having the exact same name as the ports and reside inside the current design. You cannot use a *name_patterns* list when using this option.

- **-of_gate_pins gate_pin_objects**

An optional switch and value pair that constrains the command to return the net objects associated to the `gate_pin_objects`. If a pin exists with the same name as the net, the net is returned using the command “`get_nets -of_pins`” and the “`get_nets -of_gate_pins`” command returns an empty collection. The command also returns an empty collection if the `gate_pin` is a port, in which case the command “`get_nets -of_ports`” returns the desired net. Refer to [Example 7](#) in the `get_pin` command section for an example usage.

- **-of_pseudo_ports pseudo_port_objects**

An optional switch and value pair that constrains the command to return the net objects associated to the `pseudo_port_objects`. If the pseudo port is defined on a pin, the pin is returned using the command `get_pins -of_pseudo_ports` while `get_nets -of_pseudo_ports` returns an empty collection. You can use this switch in combination with the `-silent` switch to determine if a `pseudo_port` was created on a net. Refer to [Example 6](#) in the `get_pin` command section for an example usage.

- **-filter attribute_equation**

An optional switch and string that specify to filter results based on the expression specified by the `attribute_equation` string. The attributes used within the equation must exist for the net object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The `=~` and `!~` matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the `-regexp` option or not.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for nets matching the *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the

name_patterns disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

In the following example, the collection of all nets is filtered based on an attribute_equation of user-defined attributes.

```
get_nets -filter {MyType1 || myType2 == "AB"}
{u3_Y u4_O2}
```

Example 2

In the following example, the collection of all nets matching a list of hierarchical name patterns are returned. The names u1/* and u*/u* are only searched relative to the current design. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for an example of pattern matching with regular expressions.

```
get_nets {u1/* u*/u*}
{u4/u4_Y u4/u1_Y u4/u2_Q u5/u4_Y u5/u1_Y u5/u2_Q}
```

Example 3

In the following example, the collection of all nets matching a leaf name pattern is returned. The leaf net name u4* searches for all nets found below the current design.

```
get_nets u4* -hierarchical
{u4_O1 u4_O2 u4_O3 u4/u4_Y u5/u4_Y}
```

Related Topics

[create_net](#)
[delete_pins](#)
[delete_nets](#)
[get_icl_ports](#)

[get_open_pattern_set](#)

Context: patterns -ijtag

Mode: analysis

Returns the name of the currently open pattern set.

Usage

```
get_open_pattern_set
```

Description

Returns the name of the currently open pattern set.

This command returns an empty string if no pattern set is open.

Arguments

None

Return Values

The name of the currently open pattern set.

Examples

The following example prints the name of the currently open pattern set:

```
puts "The currently opened pattern set is called [get_open_pattern_set]"
```

Related Topics

[close_pattern_set](#)

[get_pattern_set_list](#)

[open_pattern_set](#)

[report_pattern_sets](#)

[reset_open_pattern_set](#)

get_pattern_cycle_count

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.

Usage

```
get_pattern_cycle_count
```

Description

Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.

The get_pattern_cycle_count command reports the total number of cycles in the currently loaded test pattern set. The test pattern set must be STIL or WGL format.

Arguments

None

Examples

Example 1

The following example sets the pattern source to an external source, the *pat.wgl* file and displays the total number of cycles in the *pat.wgl* test pattern file:

```
read_patterns pat.wgl -wgl  
get_pattern_cycle_count
```

Example 2

The following example sets the pattern source to an external pattern database source, the *pat1.wgl* file, appends the *pat2.wgl* test pattern file to it, and displays the total number of cycles contained in both test pattern files:

```
read_patterns pat1.wgl  
read_patterns pat2.wgl --append  
get_pattern_cycle_count //reports total cycles for pat1 and pat2
```

Related Topics

[read_patterns](#)

get_pattern_set_data

Context: patterns -ijtag

Mode: analysis

Returns requested details of the internal representation of the specified pattern set.

Usage

```
get_pattern_set_data pattern_set_name [-unrolled] {-port_list | -event_list |  
-event_data event_id}
```

Description

Returns requested details of the internal representation of the specified pattern set.

The get_pattern_set_data command provides introspection into the internal representation of retargeted IJTAG patterns. Among other uses, this information can be used to generate pattern files for non-IJTAG compliant access mechanisms (for example, IEEE 1149.7), as long as the access hardware can produce the retargeted IJTAG patterns; that is, as long as the access mechanism does not put any restrictions on IJTAG retargeting.

Arguments

- ***pattern_set_name***

A required string that specifies the pattern set from which to introspect the requested information.

- **-unrolled**

An optional switch that specifies to restrict the information returned when the -event_list or -event_data switches are used. The -unrolled switch has no effect on the -port_list switch. For more information on using -unrolled with -event_list and -event_data, refer to the descriptions of those switches below.

- **-port_list**

An optional switch that specifies to return the list of ports to which values are written or from which values are read in the specified pattern set.

- **-event_list**

An optional switch that specifies to return the list of all event identifiers of the pattern set.

When the -event_list switch is used without the -unrolled switch, the returned list contains identifiers of events of type vector, scan, reset, loop, goto_state or nop.

When the -event_list switch is used with the -unrolled switch, the returned list only contains identifiers of events of type vector, loop and nop. Events of type scan, reset and goto_state are unrolled or broken into the appropriate vector events.

See “Event Types” below for details of all the event types.

- **-event_data event_id**

An optional switch and string pair that specifies to return a list of key/value pairs of event data. *event_id* must be a string from the list returned by “`get_pattern_set_data -event_list`”. You must first request the list of event identifiers from the pattern set, then you can use the event identifiers to get the pattern set data event by event.

Note

 It is your responsibility to ensure that the `-unrolled` switch is used consistently. The event identifiers obtained by means of “`get_pattern_set_data -event_list`” only make sense in “`get_pattern_set_data -event_data event_id`”, and the event identifiers obtained by means of “`get_pattern_set_data -unrolled -event_list`” only make sense in “`get_pattern_set_data -unrolled -event_data event_id`”. As the *event_ids* are not guaranteed to be mutually exclusive, you must be careful to use `-unrolled` consistently.

The return value of `get_pattern_set_data pattern_set_name -event_data event_id` is always a list with an even number of elements, in which the first, third, fifth ... elements represent keys and the second, fourth, sixth ... elements represent values. This convention allows the conversion into an associative TCL array by means of the following command:

```
array set <variable name> <even-sized list>
```

TCL allows you to iterate on a list using several variables. This is useful to iterate on the results of `get_pattern_set_data`. For example, you can do the following:

```
foreach {key value} [get_pattern_set_data test -event_data] { ... }
```

Some of the key/value pairs are guaranteed to appear in the result, others appear only if the related values exist in the retargeted patterns. The entries in the tables are flagged as “mandatory” or “optional” to reflect this.

Event Types

- A vector event represents a single parallel vector, the stimulus data of the primary inputs (or bidirectional ports) of the design and the response data of primary outputs (or bidirectional ports).

A scan event represents one scan load/unload.

A reset event represents a global reset (such as would happen, for example, by applying a ‘0’ to the TRST port of a TAP controller).

A loop event represents a certain idle time. The `vector_count` in the event data refers to tester periods, potentially rounded towards the next integer.

A `goto_state` event represents the transition from one TAP state to another one. The paths for those transitions are the same as the default paths for TAP state transitions in SVF.

A `nop` event (“no operation”) represents the absence of any action. It is only required to hold the information about notes (iNotes) that have been added after the last actual action.

The type of the event can be derived from the identifier names. An event of type vector has an identifier named vector number, where number is an increasing index. An event of type scan has an identifier named scan number and so on.

The following is an example of an event list:

```
{reset0 vector0 scan0 scan1 loop0 scan2 loop1 scan3 vector1}
```

Table 4-7. Events of type “vector”

Key	Value
port_values (mandatory)	<p>String consisting of the characters ‘U’, ‘D’, ‘0’, ‘1’, ‘X’, ‘Z’, ‘M’, ‘P’ and ‘–’. The order is the same as in the port list (returned by get_pattern_set_data -port_list)</p> <p>The meaning of the characters is as follows:</p> <ul style="list-style-type: none"> D expect 0 (hint: “down”) U expect 1 (hint: “up”) M expect Z (hint: “middle” or “median”) X do not expect any specific value L drive 0 (hint: “low”) H drive 1 (hint: “high”) Z drive Z <p>P pulse (only appears in combination with TCKPorts or capture_shift_clocks)</p> <p>– used for ScanInPorts which are currently not involved in a scan load</p> <p>D, U, X, L, H and Z are the same as in SVF. IJTAG adds P, M and – to the list of possible states</p>
write_var.port_list (optional)	Names of the primary input ports (or bidirectional ports) which the write_var.name_list and write_var.inversion_list specifications refer to. Each port can occur only once
write_var.name_list (optional)	Names of iWriteVars associated with the ports. One element per port, same order as in the write_var.port_list
write_var.inversion_list (optional)	Specifies the inversion of the iWriteVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the write_var.port_list

Table 4-7. Events of type “vector” (cont.)

Key	Value
read_var.port_list (optional)	Names of the primary output ports (or bidirectional ports) which the read_var.name_list and read_var.inversion_list specifications refer to. Each port can occur several times, because an output port can be used to observe several targets at the same time
read_var.name_list (optional)	Names of iReadVars associated with the ports. One element per port, same order as in the read_var.port_list.
read_var.inversion_list (optional)	Specifies the inversion of the iReadVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the read_var.port list
notes.text_list (optional)	List of notes preceding this event
notes.type_list (optional)	List of the types of the notes. The list has the same number of elements as notes.text_list. The elements can be either “user” or “process”. “user” notes are those which have been generated as a result of the iNote command. “process” notes are notes generated by the tool

Table 4-8. Events of type “scan”

Key	Value
port_values	All values except “–” have the same meaning as in “vector”. In events of type “scan”, the value “–” is used for <i>all</i> ScanInPorts. The actual scan load data can be found at <chain_id>.si_values. If the port_values contain expected values other than “X”, this indicates that the comparison must be done in each cycle of the scan load
write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in “vector”
scan_interface (mandatory)	Name of the scan interface (as specified in ICL)

Table 4-8. Events of type “scan” (cont.)

Key	Value
scan_type (mandatory)	“DR” or “IR”. Always “DR” for designs without TAP
start_state (mandatory)	Start state of the TAP controller. Can be one of IDLE, DRPAUSE or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, “DRPAUSE” means that there is no capture in this scan load, while “IDLE” means that there is a capture
end_state (mandatory)	End state of the TAP controller. Can be one of IDLE, DRPAUSE or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, “DRPAUSE” means that there is no update in this scan load, while “IDLE” means that there is an update
chain_list (mandatory)	List of identifiers of active chains. If the active ScanInterface does not contain Chain statements, then the list consists of only one single element, which is called “default”
<chain_id>.length (mandatory)	Length of the scan chain
<chain_id>.offset (mandatory)	The number of overshift cycles, that is the difference between the number of shift cycles and the length of the chain
<chain_id>.inversion (mandatory)	The chain inversion between ScanInPort and ScanOutPort
<chain_id>.si_values (mandatory)	Scan-in value. A string consisting of ‘0’ and ‘1’ as wide as length+offset
<chain_id>.so_values (mandatory)	Scan-out value. A string consisting of ‘0’, ‘1’ and ‘X’ as wide as length+offset
<chain_id>.write_var.cycle_list (optional)	List of cycles which the <chain_id>.write_var.name_list and <chain_id>.write_var.inversion_list refer to
<chain_id>.write_var.name_list (optional)	List of iWriteVars, consistent with cycle_list, analogously defined as write_var.name_list

Table 4-8. Events of type “scan” (cont.)

Key	Value
<chain_id>.write_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as write_var.inversion_list Note: the inversions take the scan-in inversions into account, such that it can be derived directly what has to be applied to the scan-in port
<chain_id>.read_var.cycle_list (optional)	List of cycles which the <chain_id>.read_var.name_list and <chain_id>.read_var.inversion_list refer to
<chain_id>.read_var.name_list (optional)	List of iReadVars, consistent with cycle_list, analogously defined as read_var.name_list
<chain_id>.read_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as read_var.inversion_list Note: the inversions take the scan-out inversions into account, such that it can be derived directly what has to be expected at the scan-out port
<chain_id>.register.name_list	The names of the ICL ScanRegisters in the correct order from scan-in to scan-out
<chain_id>.register.length_list	The list of register lengths in the order from scanin to scan-out. The elements in this list can be mapped uniquely to the elements in the list which is referenced by <chain_id>.register.name_list

Table 4-9. Events of type “reset”

Key	Value
notes.text_list	Same as in “vector”
notes.type_list	
sync (mandatory)	“On” or “Off”. Specifies whether a synchronous reset shall be enforced
type (mandatory)	“user” or “process”. “user” indicates the usage of iReset, whereas “process” indicates a reset generated by the tool, for example on behalf of the open_pattern_set command (if used without no_initial_ireset)

Table 4-10. Events of type “loop”

Key	Value
port_values	Same as in “vector”
write_var.port_list	If the port_values contain expected values other than ‘X’, this indicates that the comparison must be done in each cycle of the loop
write_var.name_list	
write_var.inversion_list	
notes.text_list	
notes.type_list	
vector_count (mandatory)	Duration in terms of tester periods, rounded towards the next integer

Table 4-11. Events of type “goto_state”

Key	Value
port_values	Same as in “vector”
write_var.port_list	If the port_values contain expected values other than ‘X’, this indicates that the comparison must be done in each cycle of the transition
write_var.name_list	
write_var.inversion_list	
notes.text_list	
notes.type_list	
start_state (mandatory)	IDLE, DRPAUSE or IRPAUSE
end_state (mandatory)	IDLE, DRPAUSE or IRPAUSE

Table 4-12. Events of type “nop”

Key	Value
notes.text_list	Same as in “vector”
notes.type_list	

Examples

The following example illustrates the usage of the `get_pattern_set_data` command.

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl data/icl/*.icl
SETUP> set_current_design chip
SETUP> add_clocks ClkA -period 10ns
SETUP> set_system_mode analysis
ANALYSIS> source data/pdl/raw1.iprocs
ANALYSIS> open_pattern_set test1 -tester_period 100ns
ANALYSIS> iCall block1_I1.raw1_I1.run_testa
ANALYSIS> close_pattern_set
ANALYSIS> get_pattern_set_data test1 -port_list

tck tdi tms trst tdo

ANALYSIS> get_pattern_set_data test1 -event_list
reset0 scan0 scan1 scan2 scan3 scan4 scan5 scan6 loop0 scan7

ANALYSIS> array set pd [get_pattern_set_data test1 -event_data scan5]
ANALYSIS> puts $pd(chain_list)

default

ANALYSIS> puts $pd(default.length)
16

ANALYSIS> puts $pd(default.read_var.name_list)
block1_I1.raw1_I1.go block1_I1.raw1_I1.done

ANALYSIS> puts $pd(default.read_var.cycle_list)
7 8

ANALYSIS> puts $pd(default.so_values)
XXXXXXXX00XXXXXXXX

ANALYSIS> exit -f
```

Related Topics

[open_pattern_set](#)
[close_pattern_set](#)

[get_pattern_set_list](#)

Context: patterns -ijtag

Mode: analysis

Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

Usage

```
get_pattern_set_list
```

Description

Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

The currently open pattern set is not part of the list.

Arguments

None

Return Values

A Tcl list of all existing and closed pattern sets in the order of their creation.

Examples

The following example sets the patlist variable to the set of all existing pattern sets.

```
set patlist [get_pattern_set_list]
```

Related Topics

[close_pattern_set](#)

[open_pattern_set](#)

[report_pattern_sets](#)

[reset_open_pattern_set](#)

get_pattern_set_option

Context: patterns -ijtag

Mode: analysis

Returns the value of the requested option for the specified pattern_set_name.

Usage

```
get_pattern_set_option [pattern_set_name] {-timeplate_name | -tester_period | -cycles |  
-tck_ratio | -initial_ireset | -active_scan_interfaces | -network_end_state |  
-tap_start_state | -tap_end_state | -saved}
```

Description

Returns the value of the requested option for the specified pattern_set_name.

Arguments

- ***pattern_set_name***
Specifies an optional pattern set name from which to introspect the specified option. When not specified, the currently open pattern set is introspected.
- **-timeplate_name**
Returns the name of the optional timeplate that was used when opening the pattern set. Returns null if the open_pattern_set -timeplate option was not specified; in this case the default timeplate is used. See [open_pattern_set](#) for a description of the default timeplate.
- **-tester_period**
Returns the tester period used by the pattern set.
- **-cycles**
Returns the complete number of tester cycles required to execute the pattern set on a tester.
- **-tck_ratio**
Returns the tck_ratio value of the pattern set.
- **-initial_ireset**
Returns 1 when the pattern set was opened without the -no_initial_ireset option; returns 0 when the option was used.
- **-active_scan_interfaces**
Returns the names of the active scan interfaces.
- **-network_end_state**
Returns keep, initial, or reset based on the value used with the close_pattern_set -network_end_state option.

- **-tap_start_state**
Returns the expected TAP start state. The possible values are: “IDLE”, “DRPAUSE”, “IRPAUSE” or “any”.
- **-tap_end_state**
Returns the established TAP end state. The possible values are: “IDLE”, “DRPAUSE”, or “IRPAUSE”. An error message is issued if you try to get the TAP end state of an open pattern set.
- **-saved**
Returns 1 when the pattern set was written out using at least one [write_patterns](#) command.
Returns 0 otherwise.

Return Values

The value of the requested option.

Examples

In the following example, the total number of cycles for pattern set P1 is stored in the Tcl variable V1.

```
set V1 [get_pattern_set_option P1 -cycles]
```

Related Topics

[open_pattern_set](#)
[close_pattern_set](#)

get_pins

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified *name_patterns* list and subject to filtering imposed by the specified options.

Usage

```
get_pins [name_patterns]
  [-of_instances instance_objects
   | -below_instances instance_objects
   | -of_nets net_objects
   | -of_gate_pin gate_pin_objects
   | -of_pseudo_ports pseudo_port_objects]
  [-hierarchical]
  [ -use_path_matching_options ]
  [ -filter attribute_equation]
  [ -direction input | output | inout]
  [ -regexp]
  [ -nocase]
  [ -silent]
```

Description

Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified *name_patterns* list and subject to filtering imposed by the other specified options.

When the *name_patterns* list is omitted, all pins are returned subject to filtering imposed by the specified options.

Arguments

- *name_patterns / leaf_instance/leaf_name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all pins. The name patterns are not allowed to include more than one slash which is the hierarchy separator symbol. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a complete description of the pattern syntax when the -regexp option is used and not used.

This argument cannot be specified with either the -of_nets or -of_gate_pin arguments.

- *-of_instances instance_objects*

An optional switch and value pair that constrains the command to search for pins on the specified *instance_objects*. When this option is specified, the optional *name_patterns* must be *leaf_name* patterns that are checked against the leaf name of the pins. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the

-of_instances option is used. You could traverse the *module_name* attribute of the instance objects and look up the instantiated modules but this method is more direct and efficient and automatically deals with libraries and architectures.

- **-below_instances *instance_objects***

An optional switch and value pair that constrains the command to search for pins below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the **-below_instances** switch of the [get_instances](#) command description for the justification of this behavior.

- **-of_nets *net_objects***

An optional switch and value pair that constrains the command to return the pin objects associated to *net_objects*. Those are the pins having the exact same name as the nets and reside on the instances in which the nets are in. You cannot use a *name_patterns* list when using this option.

- **-of_gate_pin *gate_pin_objects***

An optional switch and value pair that constrains the command to return the pin objects associated to the specified *gate_pin_objects*. Given, a collection or list of *gate_pin* objects, the option is used to map the *gate_pin* object to the corresponding hierarchical pin objects. Any *gate_pin* object not having an associated hierarchical pin object is discarded. You cannot use a *name_patterns* list when using this option.

- **-of_pseudo_ports *pseudo_port_objects***

An optional switch and value pair that constrains the command to return the design pins associated with the specified pseudo-ports (user-added primary inputs or outputs).

- **-hierarchical**

An optional string that specifies to perform pattern matching relative to all instances found below the current design. When this option is specified, the *name_patterns* argument cannot include more than one slash which are hierarchy separator symbols.

Using “`get_pins leaf_name_pattern -hierarchical`” is a very expensive operation. Avoid using this operation on large designs. See [Example 8](#) for a better way to achieve the same result.

- **-use_path_matching_options**

An optional switch that performs pin name-mapping using rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option allows you to find a pin using the instance mapping rules. See [add_rtl_to_gates_mapping](#) for complete information.

- **-direction input | output | inout**

An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- **-filter *attribute_equation***

An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the pin object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The $=\sim$ and $!\sim$ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the *-regexp* option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for pin names matching *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This *-silent* option has no effect on any other error reporting.

Note

 You can use the *-silent* switch to manage the case in which an empty collection is returned. Before using the *-silent* switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

In the following example, the collection of all pins is filtered based on an *attribute_equation* of user-defined attributes.

```
get_pins -filter {myType1 || myType2 == "ABC"}  
{u1/Y u4/u4/A0 u7/D}
```

Example 2

In the following example, the collection of all pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for examples of pattern matching with regular expressions.

```
get_pins {u1/Y u*/Q}
{u1/Y u2/Q u7/Q}
```

Example 3

In the following example, the pins with leaf_name matching Q* and found on instances of module DFF are returned

```
get_pins Q* -of_instances [get_instances -of_modules DFF]
{u2/reg1/Q u2/reg1/QB u3/reg5/Q}
```

Example 4

In the following example, the collection of all pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

```
get_pins u*/Y* -hierarchical
```

```
{u1/Y u3/Y u4/u1/Y u4/u4/Y u4/u5/u1/Y u5/u1/Y u5/u3/Y u5/u4/Y u6/Y}
```

Example 5

This example return the pin names of the specified pseudo_port objects:

```
get_pins -of_pseudo_ports {user_pi1 user_pi2}
{u1/Y u3/Y u4/Y}
```

Example 6

This example shows a small proc used to get the net or pin on which the pseudo_port was created:

```
proc get_objects_from_pseudo_ports {pseudo_port} {
    set objects {}
    append_to_collection objects [get_pins -of_pseudo_ports $pseudo_port
        -silent]
    append_to_collection objects [get_nets -of_pseudo_ports $pseudo_port
        -silent]
    return $objects
}
```

Example 7

This example shows a small proc used to get the net, pin, or port associated to a gate_pin:

```
proc get_objects_from_gate_pins {gate_pin} {
    set objects {}
    append_to_collection objects [get_pins -of_gate_pin $gate_pin -silent]
    append_to_collection objects [get_ports -of_gate_pin $gate_pin -of_type
        all -silent]
    append_to_collection objects [get_nets -of_gate_pin $gate_pin -silent]
    return $objects
}
```

Example 8

This example shows how to find all pins on instances of cell modules matching a given name list. One could have used the following:

```
get_pins -hierarchical {VSS VDD}
```

but this operation would be very slow on a large design given that every pin of every instance needs to be visited. Instead, use the following example:

```
foreach_in_collection cell_mod [get_modules -of_type cell] {  
    set ports [get_ports {VSS VDD} -of_mod $cell_mod -silent]  
    if {[sizeof_collection $ports] > 0} {  
        set inst [get_instances -of_modules $cell_mod -silent]  
        append_to_collection pins [get_pins {VSS VDD} -of_inst $inst -silent]  
    }  
}
```

Related Topics

[add_rtl_to_gates_mapping](#)
[create_pin](#)
[delete_pins](#)
[get_common_parent_instance](#)
[get_modules](#)
[get_nets](#)
[get_icl_ports](#)

get_ports

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical ports on a given module that match the specified *name_patterns* list and subjected to filtering imposed by the other options.

Usage

```
get_ports [name_patterns]
  [-of_modules module_objects | -of_nets net_objects | -of_pins pin_objects | -of_gate_pins
   gate_pin_objects | -of_wrapper_scan_elements scan_element_spec ]
  [-direction input | output | inout] [-of_type real | pseudo | all] [-filter attribute_equation]
  [-regexp] [-nocase] [-silent]
```

Description

Returns a collection of all hierarchical ports on a given module that match the specified *name_patterns* list and subjected to filtering imposed by the other options.

Note

 Ports are not properly returned when the module has a parameter that specifies the size of the port, but the module is not instantiated in the design.

You can have any number of wildcards or regular expressions in the name patterns.

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ports. If no *name_patterns* are specified, the tool searches all ports.

- -of_modules *module_objects*

An optional switch and value pair that constrains the command to return only the ports on the modules specified by *module_objects*.

- -of_nets *net_objects*

An optional switch and value pair that constrains the command to return the port objects associated to *net_objects*. These are the ports having the exact same name as the nets and residing on the current design. You cannot use a *name_patterns* list when using this option.

- -of_pins *pin_objects*

An optional switch and value pair that constrains the command to return the ports defined on the modules that correspond to the pins specified by *pin_objects*.

You can further constrain the command to return only [Pseudo_ports](#) or only real [Ports](#) using “-of_type pseudo | real”; if you do not specify the -of_type switch, by default, the tool returns all real ports (-of_type real). If you use this option in conjunction with “-of_type

pseudo”, the tool maps the pin that was added as part of the pseudo_port (user-added PI/PO) to the name of the associated pseudo_port. See [Example 4](#). If you added a user-specified primary input or output (pseudo_port) without specifying the pin name using “add_primary_inputs -pseudo_port_name,” the name of the pseudo_port may not always match the name of the pin specified; in this case, you can use this switch to find the name of the associated pseudo_port.

You cannot use this switch with the “-of_type all” option. You also cannot use this switch with a *name_patterns* list.

- **-of_gate_pins gate_pin_objects**

An optional switch and value pair that constrains the command to return the ports on the current design associated to the gate_pin object. Any gate_pin object not associated to a port object on the current design is discarded. You cannot use a *name_patterns* list when using this option.

- **-of_wrapper_scan_elements scan_element_spec**

An optional switch and value pair that returns a collection of port objects isolated by the supplied scan_element objects. You can use this switch for dedicated and shared wrapper cells.

Note



This switch is only available in dft -scan context in analysis mode.

- **-direction input | output | inout**

An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- **-of_type real| pseudo | all**

An optional switch and literal pair that specifies to filter results based on the specified type of the port object. When omitted, only real design ports are considered. The option types have the following meaning:

real — Only real design ports ([Port](#) object type) are included in the returned collection.

pseudo — Only user-added pseudo ports ([Pseudo_port](#) object type) are included in the returned collection.

all — Both real (design) and pseudo (user-added) ports are included in the returned collection.

- **-filter attribute_equation**

An optional switch and string that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the port object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format. The $=~$ and $!~$ matching operators use simple wildcard matching versus Posix extended regular expression based on the presence of the $-regexp$ option.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

This example looks for ports on the current design whose name match I* or O*

```
get_ports {I* O*}  
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

```
get_ports {[[:alnum:]]{1,2}} -regexp -of_modules ModA  
{ab a b}
```

Example 3

This example return ports based on the option specified for the -of_type switch:

```
add_primary_input -internal inst1/Y inst1/Z -pseudo_port_name user_pi1  
add_primary_input -internal inst2/Y -pseudo_port_name user_pi2  
get_ports  
{port1 port2 port3}  
get_ports -of_type real
```

```
{port1 port2 port3}

get_ports -of_type pseudo

{user_pi1 user_pi2}

get_ports -of_type all

{port1 port2 port3 user_pi1 user_pi2}

get_pins -of_pseudo_ports user_pi1

{inst1/Y inst1/Z}

get_pins -of_pseudo_ports user_pi2

{inst2/Y}

get_pins -of_pseudo_ports {user_pi1 user_pi2}

{inst1/Y inst1/Z inst2/Y}
```

Example 4

This example returns the pseudo_ports associated with the user-specified primary inputs and primary outputs based on the option specified for the -of_type switch:

```
add_primary_input -internal inst1/Y inst1/Z -pseudo_port_name user_pi1
add_primary_input -internal inst2/Y

get_ports -of_pins inst1/Y

{Y}

get_ports -of_pins {inst1/Y inst1/Z} -of_type real

{Y Z}

get_ports -of_pins inst1/Y -of_type pseudo

{user_pi1}

get_ports -of_pins inst2/Y -of_type real

{Y}

get_ports -of_pins inst2/Y -of_type pseudo

{inst2/Y}

get_ports -of_pins {inst1/Y inst1/Z inst2/Y} -of_type pseudo

{user_pi1 user_pi1 inst2/Y}
```

Related Topics

[create_port](#)
[delete_ports](#)

get_procfile_name

Context: all contexts

Mode: all modes

Returns the name of the procfile that was defined using the `set_profile_name` command.

Usage

```
get_procfile_name
```

Description

Returns the name of the procfile that was defined using the `set_profile_name` command.

Arguments

None

Return Values

The name of the procfile as it was specified to the `set_procfile_name` command.

Examples

In the following example, the path to the specified procfile name is stored in a Tcl variable.

```
set proc_file [get_procfile_name]
```

Related Topics

[set_procfile_name](#)

[get_read_verilog_option](#)

Context: unspecified, all contexts

Mode: all modes

Returns the options specified with the [set_read_verilog_options](#) command.

Usage

```
get_read_verilog_option -allow_enum_relaxation
```

Description

Returns the options specified with the [set_read_verilog_options](#) command.

Arguments

- `-allow_enum_relaxation`

An option that returns a 1 if the `set_read_verilog_options -allow_enum_relaxation` command and switch are set to on. A 0 is returned otherwise.

get_resource

Context: unspecified, all contexts

Mode: all modes

Returns true when all resources in the specified resource_list are available.

Usage

```
get_resource resource_list
```

Description

Returns true when all resources in the specified resource_list are available.

You use this command inside your Tcl procs to check for the availability of resources and take appropriate action when they are missing. For example, you can choose to issue the create_flat_model command if your command depends on it and it is not yet available.

Arguments

- *resource_list*

A Tcl list of strings that contains any of the following resource names:

design_is_elaborated

Used to check that the hierarchical data model has been created using the set_current_design command.

design_modules_exist

Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the get_modules command.

flat_model_exists

Used to check that the flat data model is available. The flat data model was created by either the create_flat_model command or by the “set_system_mode analysis” command.

icl_is_elaborated

Used to check that the ICL hierarchical data model exists. It exists after the set_current_design command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

icl_modules_exist

Used to check that at least one ICL module exists in memory and can be accessed using the get_icl_modules command.

rtl_mode_enabled

Used to check that the -rtl option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

Examples

The following example checks that both design and ICL modules exist.

```
if {![get_resource {design_modules_exist icl_modules_exist}]} {  
    return -code error "you must first read in design and icl modules \  
    before executing this command"  
}
```

Related Topics

[create_flat_model](#)

[read_icl](#)

[read_verilog](#)

[read_vhdl](#)

[set_context](#)

[set_current_design](#)

[set_system_mode](#)

get_run_synthesis_options

Context: unspecified

Mode: all modes

Returns the values for options that can be set for the run_synthesis command.

Usage

```
get_run_synthesis_options {[{-synthesis_output_directory]
| [synthesis_tool] [-compilation_options | -pre_compilation_commands
-post_compilation_commands | -startup_file | -command_name command_name_path]}}
[-get_dictionary]
```

Description

Returns the default values for options that can be set for [run_synthesis](#) command. The options defined with [set_run_synthesis_options](#) are overridden if they are explicitly defined with the run_synthesis command.

Arguments

- **-synthesis_output_directory**
Returns the synthesis output directory path. The default directory is *synthesis_outdir* and is located in the Tesson Shell invocation directory.
- ***synthesis_tool***
A required value that specifies the synthesis tool you wish to use for synthesis. The only permitted value is “dc_shell”. Other synthesis tools may be supported in future releases.
- **-compilation_options**
Returns the option list that adds additional options to the synthesis compile command used in the <synthesis_tool>.synthesis_tcl.
- **-pre_compilation_commands**
Returns the command list that is added before the compile command in the <synthesis_tool>.synthesis_tcl.
- **-post_compilation_commands**
Returns the command list that is added after the compile command in the <synthesis_tool>.synthesis_tcl.
- **-startup_file**
Returns file path that allows the user to explicitly specify the startup file that will be used by synthesis.
- **-command_name *command_name_path***
Defines the command name to invoke the synthesis tool. The default is dc_shell.

- **-get_dictionary**
Returns all set synthesis options as a tcl ‘dict’.

Examples

The following are examples of the `get_run_synthesis_options` using various options.

Example 1

This example returns the default `pre_compilation_commands`:

```
INSERTION> get_run_synthesis_options dc_shell -pre_compilation_commands \
           optimize_registers
```

Example 2

This example returns the default compilation options:

```
INSERTION> get_run_synthesis_options dc_shell -compilation_options \
           -map_effort high
```

Related Topics

[check_synthesis](#)
[report_run_synthesis_options](#)
[run_synthesis](#)
[set_run_synthesis_options](#)

get_scan_chain_families

Context: dft -scan

Mode: analysis

Gets scan chain families with certain characteristics.

Usage

```
get_scan_chain_families [name_patterns]  
    [ -regexp ]  
    [ -nocase ]
```

Description

Use this command to get scan chain families with certain characteristics.

Arguments

- *name_patterns*
An optional list that allows getting scan chain families by their name or name pattern.
- -regexp
An optional switch that allows getting scan chain families by regular expression matching.
- -nocase
An optional switch that ignores character casing during matching.

get_scan_chain_option

Context: pattern -scan

Mode: setup, when LBIST is on

Returns the setting of an option specified in the set_scan_chain_options command.

Usage

```
get_scan_chain_option {-chain_name chain_name... | -INInstance instance_name
                      -Block_chain_index_list block_chain_index_list...}
                      [-decompressor_to_scan_in_inversion | -scan_out_to_compactor_inversion ]
```

Description

Returns the setting of an option specified in the set_scan_chain_options command.

Arguments

- **-chain_name***chain_name*...

Required switch and repeatable string that specifies the name of a scan chain in the current design.

- **-INInstance** *instance_name* **-Block_chain_index_list** *block_chain_index_list*...

Required set of switches and values that specifies a core instance and the list of chain indices starting with “1” on the EDT controller. When this is used, the masking applied to the specified chain is OX. Use this option in conjunction with the -USed_chains switch.

- **-decompressor_to_scan_in_inversion**

An optional Boolean switch that returns 1 when the option was explicitly set with the set_scan_chain_options command; otherwise a 0 is returned.

- **-scan_out_to_compactor_inversion**

An optional Boolean switch that returns 1 when the option was explicitly set with the set_scan_chain_options command; otherwise a 0 is returned.

Examples

Suppose you specified the following command:

```
set_scan_chain_options -chain_name chain3 -decompressor_to_scan_in_inversion on
                       -scan_out_to_compactor_inversion on
```

The get_scan_chain_option would return the following values:

```
get_scan_chain_option -chain_name chain3 -decompressor_to_scan_in_inversion
```

```
1
```

```
get_scan_chain_option -chain_name chain3 -scan_out_to_compactor_inversion
```

```
1
```

get_scan_elements

Context: dft (with no sub-context), dft -scan, dft -test_points -no_rtl

Mode: setup, analysis, insertion

Returns a collection of scan elements that the tool has identified in the design.

Usage

```
get_scan_elements [name_patterns]
  [-regexp]
  [-nocase]
  [-silent]
  [-of_modules obj_spec]
  [-of_instances obj_spec | -below_instances obj_spec]
  [-of_child_scan_modes child_mode_names]
  [-of_scan_modes obj_spec]
  [-of_pins of_pin_objects]
  [-of_chain_families of_chain_families]
  [-of_gate_pins of_gate_pins]
  [-of_wrapped_ports port_spec]
  [{-below_elements elements [-flat] | -above_scan_elements elements} [-hierarchical]]
  [-class {core | standard | wrapper}]
  [-state {ignored | unusable | unusable_child_mode | unresolved_child_mode |
            usable | usable_indirectly | any_state}]
  [-type {chain | inferred | leaf_cell | segment | shift_register | transparent_leaf_cell}...]
  [-filter attribute_equation]
```

Description

Returns a collection of scan elements that the tool has identified in the design. Scan element can be a scan chain, a scan segment, a shift register, or leaf node. For more information, see [Scan Element Object Type](#).

Arguments

- *name_patterns*
An optional list that allows getting scan elements by their name or name pattern.
- *-regexp*
An optional switch that allows getting scan elements by regular expression matching.
- *-nocase*
An optional switch that ignores character casing during matching.
- *-of_modules obj_spec*
An optional switch and value pair that gets the scan elements that are on and below all the instances of the given module(s).

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a possible mistake when specifying patterns. This **-silent** option has no effect on any other error reporting.

Note

 You can use the **-silent** switch to manage the case in which an empty collection is returned. Before using the **-silent** switch for this purpose, refer to the [set_tcl_shell_options -change_no_result_warnings_to_errors](#) switch description for detailed usage.

- **-of_instances *obj_spec***

An optional switch and value pair that gets the scan elements that are on the given instance(s).

- **-below_instances *obj_spec***

An optional switch and value pair that gets the scan elements that are below the given instance(s).

If the specified *obj_spec* is an empty collection, all instances below the current design are considered. See the description of the [-below_instances switch](#) of the [get_instances](#) command description for the justification of this behavior

- **-below_elements *elements***

An optional switch and value pair that gets the scan elements that are inside the given hierarchical scan element(s).

- **-above_scan_elements *elements***

An optional switch and value pair that gets the parent scan elements of the scan element(s).

- **-of_child_scan_modes *child_mode_names***

An optional switch and value pair that gets the existing segments declared with the given mode(s).

- **-of_scan_modes *obj_spec***

An optional switch and value pair that gets the scan elements that are included in the given mode(s), previously added with the command [add_scan_mode](#). This switch is only meaningful after issuing the [analyze_scan_chains](#) command.

- **-of_pins *of_pin_objects***

An optional switch and value pair that gets the `scan_elements` associated with the specified instance pins. Specified pins can be pins on scan element leaves or existing segment connection pins.

- **-of_chain_families** *of_chain_families*

An optional switch and value pair that gets the scan elements that are included in the given chain families, previously added with the command [create_scan_chain_family](#).

- **-of_gate_pins** *of_gate_pins*

An optional switch and value pair that gets the scan elements that contain the specified flat model gate pin(s).

- **-of_wrapped_ports** *port_spec*

An optional switch and value pair that gets a collection of wrapper scan_elements that isolate the supplied ports. You can use this switch for dedicated and shared wrapper cells.

Note

 This switch is only available in dft -scan context in analysis mode.

- **-flat**

An optional switch that recursively traverses the local hierarchies and returns only the leaf scan elements below the parent(s).

- **-hierarchical**

An optional switch that recursively traverses the local hierarchies and returns every encountered scan elements below the parent(s).

- **-class** {core | standard | wrapper}

An optional switch that returns only the scan elements of the specified class.

The possible classes are:

standard — Cell class of all elements when no wrapper analysis has been performed.

core — Cell class of all internal elements once the wrapper analysis has been performed.

wrapper — Cell class of all external elements once the wrapper analysis has been performed.

- **-state**

An optional switch that returns only the scan elements of the specified state. By default, only usable scan elements are returned.

The possible states are:

ignored — Scan elements ignored for scan insertion purposes.

unusable — Scan elements that cannot be specified in a population for distribution purposes.

unusable_child_mode — Scan elements that cannot be specified in a population for distribution purposes because the necessary “active_child_scan_mode” attribute is not set appropriately.

unresolved_child_mode — Scan elements that cannot be specified in a population for distribution purposes because the “active_child_scan_mode” was not set and cannot be inferred.

usable — Scan elements that can be specified in a population for distribution purposes.

usable_indirectly — Usable scan elements that cannot be specified directly in a population for distribution purposes because they are part of a usable hierarchical container, which must be used instead.

- **-type**

An optional, repeatable switch that returns only the scan elements of the specified type(s).

The possible types are:

chain — Existing chains and new allocated chains that were created by distribution.

inferred — Hierarchical scan elements allocated by the tool during distribution.

leaf_cell — Scan elements from a cell library.

segment — Existing segments or new virtual segment created by the user.

shift_register — Shift-register scan arrangement identified by the tool.

transparent_leaf_cell — Transparent cells (for example, lockup cell) traced inside existing chains or segments that have a scan length of zero.

- **-filter *attribute_equation***

An optional switch and value pair that returns only the scan elements that satisfy the provided attribute equation.

get_scan_modes

Contest: dft -scan

Mode: analysis

Gets scan elements with certain characteristics.

Usage

```
get_scan_modes [name_patterns]
```

Description

Use this command to get scan elements with certain characteristics.

Arguments

- *name_patterns*

An optional list that retrieves all scan elements by their name.

get_scratch_directory

Context: unspecified, all contexts

Mode: all modes

Returns the absolute path to the *scratch* directory.

Usage

```
get_scratch_directory
```

Description

Returns the absolute path to the *scratch* directory.

The tool creates the scratch directory in the temporary *.tessent.tmp.hostname.process_id* directory during tool invocation and deletes it when the tool exits. By default, it stores the scratch directory in the location identified by the TMPDIR environment variable. If you have not defined the TMPDIR environment variable, the tool creates the scratch directory in the */tmp* directory or in the current directory if the */tmp* directory is not available.

You can explicitly specify the location where the temporary *.tessent.tmp.hostname.process_id* directory gets created by setting the TESSENT_TMP_LOCATION environment variable. The tool will create the temporary *.tessent.tmp.hostname.process_id* directory in the default location if the location specified by the TESSENT_TMP_LOCATION environment variable does not exist, is not a valid directory, or does not have write permission. The tool uses the TESSENT_TMP_LOCATION environment variable when both TMPDIR and TESSENT_TMP_LOCATION exist.

Arguments

None

Examples

Suppose TMPDIR points to */home/mydir*. The following example reports the location of the scratch directory:

```
get_scratch_directory  
/home/mydir/.tessent.tmp/scratch
```

[get_silicon_insight_job_status](#)

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns the status of diagnosis jobs that are running in the background.

Usage

```
get_silicon_insight_job_status [-job_handle job_handle] [-display {active | all | completed}]  
[-message_type {screen_only | info}]
```

Description

You can use this command when running diagnosis on an ATPG instrument and have specified off for the wait_for_diagnosis_result configuration debug option.

Arguments

- **-job_handle *job_handle***

An optional switch and string pair that specifies to return the “Running” or “Completed” status of the specified job. The specified job handle is the job ID that the tool uses when it submits a job.

- **-display {active | all | completed}**

An optional switch and string pair that specifies the status type to display as a list. The choices are:

all — displays all jobs.

active — displays the actively running jobs.

complete — displays the completed jobs.

- **-message_type {screen_only | info}**

An optional switch and string pair that specifies to display the status only on-screen (the default) or both on-screen and in the transcript.

Examples

```
get_silicon_insight_job_status -display active
```

```
JOB: 'all_cores_c1_2.3/job_script.sh' completed.
```

get_silicon_insight_option

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns current settings of options specified in the set_silicon_insight_option command.

Usage

```
get_silicon_insight_option [-cdp_verification {on | off}] [-interactive_ijtag] [-current_cdp]
                           [-simdut_port] [-ignore_uncontacted_pin_data {off | on}] [-interactive_ijtag_pattern_type
                           {svf | stil}] [-maximum_failing_cycles number]
```

Arguments

- **-cdp_verification { on | off }**

An optional switch that specifies whether to verify that new test patterns added as part of a test were generated by Mentor Graphics tools, which means they contain a signature that links them to the design they were generated for. The default is “on.” You may want to turn CDP verification off if you are adding test patterns that were not generated by Mentor Graphics tools.

- **-interactive_ijtag**

An optional switch that returns “on” or “off”.

- **-current_cdp**

An optional switch that returns the current CDP.

- **-simdut_port**

An optional switch that returns the name of the SimDUT port.

- **-ignore_uncontacted_pin_data {off | on}**

An option switch and string pair that specifies whether to display or ignore uncontacted pin data.

- **-interactive_ijtag_pattern_type {svf | stil}**

An optional switch and string pair that returns either the SVF or STIL pattern type.

- **-maximum_failing_cycles number**

An optional string and integer that specifies to display the specified number of maximum failing cycles.

get_simulation_context_list

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns the available simulation contexts in a Tcl list.

Usage

```
get_simulation_context_list [-predefined | -user_defined]
```

Description

Returns the available simulation contexts in a Tcl list.

The `get_simulation_context_list` command returns all simulation contexts or returns only predefined or user-defined. If you issue the command without arguments, the command returns all simulation contexts. Note that for predefined simulation contexts, this command returns only the simulation contexts for which simulation data currently exists. This requires that the tool first read a procedure file.

Arguments

- `-predefined`
An optional switch that specifies predefined simulation contexts, which are listed in [Table 6-7](#) on page 2023.
- `-user_defined`
An optional switch that specifies user-defined simulation contexts.

Examples

The following example shows the Tcl list of values returned by the `get_simulation_context_list` command:

```

ANALYSIS> set_current_simulation_context stable_load_unload
ANALYSIS> add_simulation_context myContext1
ANALYSIS> add_simulation_context myContext2
ANALYSIS> get_simulation_context_list
{ stable_after_setup stable_load_unload stable_capture stable_shift
myContext1 myContext2 }

ANALYSIS > get_simulation_context_list -user_defined
{ myContext1 myContext2 }

ANALYSIS > get_simulation_context_list -predefined
{ stable_after_setup stable_load_unload stable_capture stable_shift }

```

Related Topics

[add_simulation_context](#)
[add_simulation_forces](#)
[report_simulation_contexts](#)
[copy_simulation_context](#)
[report_simulation_forces](#)
[delete_simulation_contexts](#)
[simulate_clock_pulses](#)
[delete_simulation_forces](#)
[simulate_forces](#)
[get_current_simulation_context](#)
[set_current_simulation_context](#)

get_simulation_library_sources

Context: unspecified, all contexts

Mode: all modes

Returns the pathnames or file extensions previously specified with the [set_simulation_library_sources](#) command.

Usage

```
get_simulation_library_sources [-path_list] [-extension_list] [-sv_extension_list]
    [-logical_library_map_list] [-include_directory_list] [-f_files]
```

Description

Returns the pathnames or file extensions previously specified with the [set_simulation_library_sources](#) command.

By default, the `get_simulation_library_sources` command returns a list of paired values that specify the pathnames to file and directory names. The first item in the pair is `-v` or `-y` to indicate whether the pathname is a file or a directory, respectively. This is useful when you want to add additional simulation library sources since any invocation of the `set_simulation_library_sources` command overwrites previous invocation settings.

Arguments

- `-path_list`

An optional switch that specifies to return the list of simulation library source paths with directory and file names. Each name is associated with a `-y` and `-v` element to identify whether the path is a directory or file name. This switch is inferred when no other switch is specified.

You can modify this returned list and feed it back to the `set_simulation_library_sources` command as shown below. The use of the `{*}` syntax is a Tcl 8.5 feature that converts the returned lists as string arguments.

```
set_simulation_library_source {*} [get_design_source -path_list] \
    {*} [get_design_source -ext_list]
```

- `-extension_list`

A Boolean switch that specifies to return the list of extensions that was used to search the library directory for the specified format.

- `-sv_extension_list`

A Boolean switch that specifies to return the list of SV extensions that was specified using the [set_simulation_library_sources](#) command.

- `-logical_library_map_list`

A Boolean switch that specifies to return the list of library directory pairs that was specified using the [set_simulation_library_sources](#) command.

- **-include_directory_list**

A Boolean switch that specifies to return the list of `include file directory pairs that were specified with the [set_simulation_library_sources](#) command.

- **-f_files**

A Boolean switch that specifies to return a list of one or more ASCII files that contain pointers to design files that were specified using the [set_simulation_library_sources](#) command.

Examples

This example specifies information about simulation library sources, and then introspects the list of pathnames and the list of extensions specified:

```
SETUP> set_simulation_library_sources -v lib1.v lib2.v -y /u/joe/lib3 -extensions .v  
SETUP> get_simulation_library_sources -path_list  
-v lib1.v -v lib2.v -y /u/joe/lib3  
  
SETUP> get_simulation_library_sources -extension_list  
.v .v.gz
```

Related Topics

[report_simulation_library_sources](#)

[set_simulation_library_sources](#)

get_simulation_option

Context: all contexts

Mode: all modes

Returns the values for options that can be set for the `set_simulation_options` command.

Usage

```
get_simulation_option
  -set_reset_dominant_port
  | -c6_mask_races
  | -multicycle_fault_simulation
  | -mux_select_x_sim_x
  | -feedback_buffer_x_init_value
  | -report_x_capturing_cells
  | -simulate_learned_sequential_tie_gates
```

Description

Returns the values for options that can be set for the `set_simulation_options` command.

This command takes one argument at a time and returns a 0 or a 1.

Arguments

- `-set_reset_dominant_port`

A switch that instructs the tool to return the `set_simulation_options` command setting of the Set/Reset port as the dominate synchronous port for both flip-flops and latches when multiple ports are on simultaneously option. Possible return values are 0 (disabled) or 1 (enabled).

- `-c6_mask_races`

A switch that instructs the tool to return the `set_simulation_options` command setting of the pessimistic simulation on C6 violated state elements and mask the capture values only when the data port transitions at the same frame the clock port captures option. Possible return values are 0 (off) or a 1 (on).

- `-multicycle_fault_simulation`

A switch that instructs the tool to return the `set_simulation_options` command setting of the multi-cycle path ATPG and fault simulation option. Possible return values are 0 (off) or a 1 (on).

- `-mux_select_x_sim_x`

A switch that instructs the tool to return the `set_simulation_options` command setting of the model_mux primitives in the design as either consensus or non-consensus option. Possible return values are 0 (off) or a 1 (on).

- **-feedback_buffer_x_init_value**

A switch that instructs the tool to return the set_simulation_options command setting of the FB_BUF primitive to carry over its value from the previous cycle instead of being initialized to X option. Possible return values are 0 (off) or a 1 (on).

- **-report_x_capturing_cells**

A switch that instructs the tool to return the set_simulation_options command setting of the option that reports scan cells that capture X during simulation. Possible return values are 0 (off) or a 1 (on).

- **-simulate_learned_sequential_tie_gates**

A switch that instructs the tool to return the set_simulation_options command setting of the change the ATPG simulator behavior for sequential gates learned as tied during DRC option. Possible return values are 0 (off) or a 1 (on).

Related Topics

[set_simulation_options](#)

[get_simulation_value_list](#)

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns a Tcl list of the simulation values on the specified gate_pin objects.

Usage

`get_simulation_value_list obj_spec`

Description

Returns a Tcl list of the simulation values on the specified gate_pin objects.

The `get_simulation_value_list` command returns the simulation values on the specified gate_pin objects in the current simulation context. The simulation value of a single gate_pin is either X, 0, 1, or Z.

Arguments

- `obj_spec`

A required value that specifies one or more gate_pin objects or pins that map to gate_pin objects (on library cell boundaries or preserved design boundaries).

Examples

The following example shows that the `get_simulation_value_list` command is equivalent with querying the `simulation_value` attribute. However, note that `get_attribute_value_list` command requires a Tcl list of gate_pins in order to work correctly, as shown in the example:

```

set_current_simulation_context stable_after_setup
get_simulation_value_list {pi clk an_2/A0 an_2/Y}
1 X 1 X

get_attribute_value_list [get_gate_pins {pi clk an_2/A0 an_2/Y}] -name simulation_value
1 X 1 X

```

Related Topics

[add_simulation_context](#)
[get_simulation_context_list](#)
[add_simulation_forces](#)
[report_simulation_contexts](#)
[copy_simulation_context](#)
[report_simulation_forces](#)

[delete_simulation_contexts](#)
[simulate_clock_pulses](#)
[delete_simulation_forces](#)
[simulate_forces](#)

get_single_attribute_value

Context: unspecified, all contexts

Mode: all modes

Retrieves the value of an attribute on the specified design object.

Usage

```
get_single_attribute_value object_spec -name attribute_name [-is_specified]
```

Description

This command is the same as the [get_attribute_value_list](#) command except the *object_spec* must be a collection with exactly one element, otherwise the tool issues an error.

The return value is a single value.

Arguments

- ***object_spec***

A required value that specifies a collection of exactly one element.

- **-name *attribute_name***

A required switch and string pair that specify the name of the attribute whose value is to be returned.

If the attribute *attribute_name* is not available for the *object_type* of *object_spec*, that is, the attribute is not registered for this *object_type* and cannot be inherited, then the tool issues an error.

- **-is_specified**

An optional Boolean switch that returns 1 when the attribute was explicitly set on the object otherwise a 0 is returned

Examples

Example 1

The following example:

```
get_single_attribute_value u1/DEF -name parent_instance
```

Is equivalent to the following [get_attribute_value_list](#) operation:

```
lindex [get_attribute_value_list u1/DEF -name parent_instance] 0
```

Example 2

The following example returns the value of *parent_instance* value for pins:

```
get_single_attribute_value u1/u5/u8/ABC -name parent_instance
```

```
u1/u5
```

Example 3

The following example returns pin names and their direction:

```
foreach_in_collection pjn [get_pins u1/*] {  
    puts "The pin [get_single_name $pin] has the direction [get_single_attribute_value \  
          $pin -name direction]."  
}
```

```
The pin u1/a has the direction input.  
The pin u1/b has the direction input.  
The pin u1/z has the direction output.
```

get_single_name

Context: unspecified, all contexts

Mode: all modes

Returns a string with the name of the element specified by the `object_spec` argument.

Usage

```
get_single_name object_spec [-remove_escaping] [-silent]
```

Description

Returns a string with the name of the element specified by the `object_spec` argument.

The `-remove_escaping` argument removes the \ and the trailing spaces in an escaped identifier. Many legacy commands such as `add_display_instance` require the instance name to have the escape character stripped from its name. However, these legacy commands do accept collections as inputs in cases where the name format is not important. See the examples for an illustration of this behavior

If the object in `object_spec` does not exist, an error message displays unless the `-silent` switch is specified. If more than one object exists in `object_spec`, an error message displays unless the `-silent` switch is specified.

Arguments

- **`object_spec`**
A required string that specifies an object name or a single-element collection.
- **`-remove_escaping`**
An optional switch that specifies to strip out the \ and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See [Example 3](#) for an illustration of this usage.
- **`-silent`**
An optional switch that specifies to suppress error messages if the object in `object_spec` does not exist.

Examples

Example 1

The following example prints out all of the modules in the database:

```
foreach_in_collection elem [get_modules] {  
    puts "Module: [get_single_name $elem]"  
}
```

```
Module: modb
Module: modc
Module: modd
Module: top
Module: GND
...
```

Example 2

The following example generates an error because the collection of pins returned by the `get_pins` commands contains more than one element.

```
get_single_name [get_pins u1/i*]
// Error: Collection contains more than one element.
```

Example 3

The following example uses the `-remove_escaping` argument to strip out the slash “\” from an escaped identifier.

```
get_single_name {\x[0] } -remove_escaping
\x[0]
```

Related Topics

[get_attribute_list](#)
[get_attribute_option](#)
[get_attribute_value_list](#)
[get_common_parent_instance](#)
[get_modules](#)
[get_icl_ports](#)
[get_name_list](#)

get_static_dft_signal_icall

Context: dft patterns

Mode: setup, analysis, insertion

Returns the iCall corresponding to the setting specified with the [set_static_dft_signal_values](#) command.

Usage

```
get_static_dft_signal_icall
```

Description

Returns the iCall corresponding to the setting specified with the [set_static_dft_signal_values](#) command.

When in dft -scan, dft -test_points, dft -edt and in patterns -scan, this command is automatically called when going from setup to analysis mode and the resulting value is inserted as if you had used it as the first “[set_test_setup_icall](#) [[get_static_dft_signal_icall](#)]” call. The command exists as a stand alone command for when you want to create a stand-alone pattern set in patterns context as shown in the example below.

This command is also called during process_patterns_specification to obtain the iCall corresponding to the DftControlSettings specified in the [Patterns](#) wrapper.

Arguments

None

Examples

The following example shows the use of the [set_static_dft_signal_values](#) followed by [report_static_dft_signal_settings](#) and the [get_static_dft_signal_icall](#) inside an [open_pattern_set](#). See the description of the [set_static_dft_signal_values](#) command to understand where the “Set value” of the DFT signals showing a “Set source” value of “Inferred” came from.

As described above, this command is automatically invoked to configure test_setup when in a context that supports the [set_test_setup_icall](#) command.

```
set_context patterns -ijtag
read_icl ./sub_block_I3.icl
set_current_design sub_block_I3

set_system_mode analysis
open_pattern_set P1
report_static_dft_signal_settings
```

```
// ICL Module      : sub_block_13
// ----- -----
// DFT Signal Name Usage          Set value  Set source
// ----- -----
// all_test         global_dft_control   -        -
// ltest_en         logic_test_control    -        -
// int_ltest_en    logic_test_control    -        -
// ext_ltest_en    logic_test_control    -        -
// int_edt         scan_mode(internal)   -        -
// ext_multi       scan_mode(external)   -        -
//
set_static_dft_signal_values int_edt 1
report_static_dft_signal_settings

// ICL Module      : sub_block_13
// ----- -----
// DFT Signal Name Usage          Set value  Set source
// ----- -----
// all_test         global_dft_control   1        Inferred
// ltest_en         logic_test_control    1        Inferred
// int_ltest_en    logic_test_control    1        Inferred
// ext_ltest_en    logic_test_control    -        -
// int_edt         scan_mode(internal)   1        Explicit
// ext_multi       scan_mode(external)   -        -
//
puts [get_static_dft_signal_icall]

dft_signal_iproc
  sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.all_test 1
  sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.int_edt 1
  sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.int_ltest_en 1
  sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.ltest_en 1

iCall {*}[get_static_dft_signal_icall]
close_pattern_set
write_patterns p1.v -verbose
```

Related Topics

[add_dft_signals](#)
[reset_static_dft_signal_values](#)
[report_static_dft_signal_settings](#)
[set_static_dft_signal_values](#)

[get_synchronous_clock_groups](#)

Context: dft, patterns

Mode: setup, analysis

Returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.

Usage

```
get_synchronous_clock_groups
```

Description

This command returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.

If no sync groups are defined, this command returns an empty list.

Arguments

None

Examples

This example creates two synchronous clock groups and uses this command in Tcl code to return the clock groups.

```
SETUP> add_synchronous_clock_group {clk1 clk2}
SETUP> add_synchronous_clock_group {clk3 clk4}
foreach grp [get_synchronous_clock_groups] {
    puts [lsort [get_name_list $grp]]
}
clk1 clk2clk3 clk4
```

Related Topics

[add_synchronous_clock_group](#)

[delete_synchronous_clock_group](#)

get_system_mode

Context: unspecified, all contexts

Mode: all modes

Returns the system mode.

Usage

```
get_system_mode [-previous | -is_in_transition]
```

Description

Returns the system mode.

Possible returned values are: setup, analysis, and insertion.

Arguments

- **-previous**
An optional switch that specifies to return the previous system mode.
- **-is_in_transition**
An optional switch that returns “1” when called within a callback that is running inside the system mode transition.

Examples

The following example returns the current system mode and the previous system mode. This enables you to know that you came to the insertion mode through the analysis mode.

```
get_system_mode
```

```
insertion
```

```
get_system_mode -previous
```

```
analysis
```

The following example shows how to use the **-is_in_transition** switch inside a callback to skip the operation when the feature is called within a system mode transition.

```
if {[get_system_mode -is_in_transition]}{
    return
}
```

Related Topics

[set_system_mode](#)

[get_tcl_shell_option](#)

Context: all contexts

Mode: setup

Introspects the Tcl shell options as specified with the [set_tcl_shell_options](#) command.

Usage

```
get_tcl_shell_option {-abort_dofile_on_error | -legacy_dofile_comments |  
                     -legacy_commands_with_spaces | -minimum_command_typing |  
                     -change_no_result_warnings_to_errors}
```

Description

Introspects the Tcl shell options as specified with the [set_tcl_shell_options](#) command. You can only specify one option at a time. The tool generates an error if you specify more than one option in a single command execution.

Arguments

- [-abort_dofile_on_error](#)
An optional switch that returns the current setting of the `abort_dofile_on_error` option. Possible return values are off, on, and exit.
- [-legacy_dofile_comments](#)
An optional switch that returns the current setting of the `legacy_dofile_comments` option. Possible return values are off and on.
- [-legacy_commands_with_spaces](#)
An optional switch that returns the current setting of the `legacy_commands_with_spaces` option. Possible return values are off and on.
- [-minimum_command_typing](#)
An optional switch that returns the current setting of the `minimum_command_typing` option. Possible return values are off and on.
- [-change_no_result_warnings_to_errors](#)
An optional switch that returns the current setting of the `change_no_result_warnings_to_errors` option. Possible return values are off and on.

Related Topics

[report_tcl_shell_options](#)

[set_tcl_shell_options](#)

[get_test_end_icall_list](#)

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: all modes

Returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list containing the iProc and its arguments.

Usage

```
get_test_end_icall_list
```

Description

Returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list containing the iProc and its arguments.

For each iCall added to the test_end procedure by the set_test_end_icall command, the get_test_end_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Examples

The following example returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list consisting of the ICLinstance path, the name of the iProc, and the arguments to the iProc.

```
set_test_end_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on \
edt_bypass off"
set_test_end_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on \
edt_bypass off" -append
get_test_end_icall
{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Topics

[iCall](#)

[set_test_end_icall](#)

[set_test_setup_icall](#)

[get_test_point_type](#)

Context: dft -test_points

Mode: setup, analysis

Reports the current test point type as specified by the [set_test_point_types](#) command.

Usage

```
get_test_point_type
```

Description

Reports the current test point type as specified by the [set_test_point_types](#) command.

The default test point type is `edt_pattern_count`.

Arguments

None.

Examples

The following example reports the default test point type, updates the test point type to `lbist_test_coverage`, and then reports the updated test point type for your design:

```
set_context dft -test_points -no_rtl
read_verilog design_name
read_cell_library cell_library_name
read_sdc sdc_file_name
get_test_point_type
edt_pattern_count

set_test_point_type lbist_test_coverage
get_test_point_type
lbist_test_coverage
```

Related Topics

[add_control_points](#)

[add_observe_points](#)

[set_test_point_types](#)

get_test_points

Context: dft -test_points

Mode: all modes

Returns a collection of all test points, or a collection of test points matching an optional Tcl list of specified test point attributes.

Usage

```
get_test_points [name_patterns]
  [-type { control | observe }] [ -origin { design_analysis | user_specified } ]
  [ -control_point_type { AND | OR }] [ -below_instances below_instances ]
  [ -of_pins of_pins] [ -filter filter_expression ]
  [ -regexp ] [ -nocase ] [ -silent ]
```

Description

This command returns a collection of all test points, or a collection of test points matching an optional Tcl list of test point name/location strings, with the option of filtering test points according to the following attributes:

- name/location (including regular expressions)
- type (control or observation)
- origin (user specified or tool generated by design analysis)
- control_point_type (AND/OR) (relevant only for control points)
- attribute values
- instance
- associated pins

Arguments

- *name_patterns*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of test points according to their location (or name). The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no location_patterns are specified, the tool searches all test points. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” for a complete description of the pattern syntax when the -regexp option is used or not used.

- **-type { control | observe }**

An optional switch and literal pair that specifies to filter results based on the specified type of the test point. When omitted, all test points are considered. The option types have the following meaning:

- control —Test points that are for the purpose of increasing controllability in design areas where, previously, the tool could not easily force certain state values. Only these test points will be included in the returned collection.
- observe —Test points that are for the purpose of increasing observability in design areas where, previously, the tool could not easily observe a certain state. Only these test points will be included in the returned collection.

- **-origin { design_analysis | user_specified }**

An optional switch and literal pair that specifies to filter results based on the specified origin of the test point. When omitted, all test points are considered. The origin options have the following meaning:

- design_analysis — Test points that were generated by the tool. Only these test points will be included in the returned collection.
- user_specified — Test points that were manually specified. Only these test points will be included in the returned collection.

- **-control_point_type { AND | OR }**

An optional switch and literal pair that enable filtering results based on the type of gate used in the control type test point. Setting this flag will filter out all test points that are not of the specified gate type (including all observe points).

- **-below_instances *below_instances***

An optional switch and string pair that returns the test points that are below the specified instance.

- **-of_pins *of_pins***

An optional switch and string pair that returns the test points that are on the specified pins.

- **-filter *filter_expression***

An optional switch and string pair that returns only the test points that satisfy the provided attribute equation. See “[Test Point Data Model](#)” for the list of Test point attributes that are valid attribute names.

- **-regexp**

An optional switch that specifies to the tool to interpret the value of the name_patterns argument as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” for a complete description of the regular expression syntax.

- **-nocase**

An optional switch that specifies to the tool to perform case-insensitive pattern matching when looking for nets matching the name_patterns.

- **-silent**

An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a possible mistake when specifying patterns. This -silent option has no effect on any other error reporting.

Note

 You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -[change_no_result_warnings_to_errors](#) switch description for detailed usage.

Examples

Example 1

The following example returns all test points in the design:

```
get_test_points
{tp#test_module2/neg1/Z tp#test_module2/neg2/Z
tp#1/test_module2/neg1/Z tp#1/test_module2/neg2/Z
tp#test_module1/neg1/Z tp#test_module1/neg2/Z
tp#1/test_module1/neg1/Z tp#1/test_module1/neg2/Z
tp#u1/U15/Z tp#u1/U14/Z tp#u1/U75/Z tp#U64/Z
tp#u1/U71/Z tp#U114/Z tp#u1/U45/Z tp#u1/U61/Z
tp#U111/Z tp#u1/U80/Z tp#u1/U42/Z tp#u1/U70/Z
tp#U119/Z tp#u1/U22/Z tp#U128/Z tp#u1/U43/Z
tp#u1/U39/Z tp#U75/Z tp#u1/U23/Z tp#U93/Z
tp#test_module2/nand2/Z tp#u1/U25/Z tp#U109/Z
tp#u1/U17/Z tp#U98/Z tp#U81/Z tp#U92/Z tp#U104/Z
tp#u1/my_module/nand2/Z tp#u1/U31/Z tp#u1/U32/Z
tp#U73/Z tp#U130/Z tp#U131/Z tp#u1/U67/Z
tp#u1/U49/Z tp#U126/Z tp#U65/Z tp#u1/U47/Z
tp#U100/Z tp#u1/U81/Z tp#u1/U53/Z ...}
```

Example 2

The following example returns all control points below the specified instances:

```
get_test_points -below_instances test_module* -type control
```

```
{tp#test_module2/neg1/Z
tp#test_module2/neg2/Z
tp#test_module1/neg1/Z
tp#test_module1/neg2/Z
tp#test_module1/neg2/A}
```

Example 3

The following example generates a report of the test points returned by the `get_test_points` command:

```
report_test_points [get_test_points -below_instances test_module* -type control]
```

```
=====
name          type      control_point_type  enable_connection_name  clock_domain
flop_location
-----
tp#test_module2/neg1/Z  control  AND           /lbist_en            clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_i

tp#test_module2/neg2/Z  control  OR           /lbist_en            clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i

tp#test_module1/neg1/Z  control  AND           /lbist_en            clk2
/test_module1/ts_default_PD_u11_cp_0sffp1_i

tp#test_module1/neg2/Z  control  OR           /lbist_en            clk2
/test_module1/ts_default_PD_u11_cp_4sffp1_i

tp#test_module1/neg2/A  control  OR           /lbist_en            clk1
/test_module1/ts_default_PD_u11_cp_2sffp1_i
```

[get_test_setup_icall_list](#)

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: all modes

Returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list containing the iProc and its arguments.

Usage

```
get_test_setup_icall_list
```

Description

Returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list containing the iProc and its arguments.

For each iCall added to the test_setup procedure by the set_test_setup_icall command, the get_test_setup_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Examples

The following example returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list consisting of the ICLinstance path, the name of the iProc, and the arguments to the iProc.

```
set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on \
edt_bypass off"
set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on \
edt_bypass off" -append
get_test_setup_icall

{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Topics

[get_test_end_icall_list](#)

[iCall](#)

[set_test_end_icall](#)

[set_test_setup_icall](#)

get_testbench_simulation_options

context: all contexts

mode: all modes

Returns the value of a given option which may have been configured using the [set_testbench_simulation_options](#) command.

Usage

```
get_testbench_simulation_options
  {[-default_simulator | -simulation_output_directory
    | -parallel_simulations ]
  | [-keep_simulation_data ]
  | [-store_simulation_waveforms ]
  | [-simulation_timeout ]
  | [ {questa | vcs} [ -simulator_options | -simulation_run_commands ]
    | -waveform_configuration_commands ] ] }
  [ -get_dictionary ]
```

Description

Returns the value of a given option that may have been configured using the [set_testbench_simulation_options](#) command.

Arguments

- [-default_simulator](#)

A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.

- [-simulation_output_directory](#)

A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. This default value is also used by the [check_testbench_simulations](#) -simulation_output_directory_list option. The default value is configurable using the [set_testbench_simulation_options](#) command.

- [-parallel_simulations](#)

A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.

- [-keep_simulation_data](#)

A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.

- **-store_simulation_waveforms**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.
- **-simulation_timeout**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.
- **questa | vcs**
A literal that specifies the simulator you want to use for simulation. The default value is “questa” from Mentor Graphics.
- **-simulator_options**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.
- **-simulation_run_commands**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.
- **-waveform_configuration_commands**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.
- **-get_dictionary**
A switch that specifies to return the value of the [run_testbench_simulations](#) command option having the same name. The default value is configurable using the [set_testbench_simulation_options](#) command.

Examples

This example returns the default value used for the -simulation_output_directory option.

```
get_testbench_simulation_option -simulation_output_directory  
./simulation_outdir
```

get_timeplate_list

Context: all contexts

Mode: all modes

Returns a list of the currently defined timeplates.

Usage

```
get_timeplate_list
```

Description

Returns a list of the currently defined timeplates. The data becomes available when you transition from setup mode to analysis mode. In analysis mode, you can change the data with subsequent [read_procfile](#) commands.

In addition to being available in analysis mode, the `get_timeplate_list` command is available in setup mode for cases when the transition to analysis mode fails after reading the procedure file, due to a DRC violation. Even though you are still in setup mode, the procedure file information is still in memory so that report and introspection commands can still be used on the information that was read.

Arguments

None

Return Values

A list of the currently defined timeplates.

Examples

```
get_timeplate_list
```

```
tp1 tp2 tp3
```

get_tool_info

Context: unspecified, all contexts

Mode: all modes

Returns information about the current tool and the current run as a Tcl result.

Usage

```
get_tool_info -name | -version | -cpu_time [ms | s | m | h | d] | -memory [b | kb | mb | gb] |
    -hostname | -interactive
```

Description

Returns information about the current tool and the current run as a Tcl result.

You must specify exactly one switch.

Arguments

- **-name**

A switch that specifies to return the name of the current tool.

- **-version**

A switch that specifies to return the version of the current tool.

- **-cpu_time [ms | s | m | h | d]**

A switch that specifies to return the user time of the process. By default, this command returns the time in seconds, but you can optionally specify one of the other units. The possible units are:

ms — Specifies the time unit is milliseconds.

s — Specifies the time unit is seconds.

m — Specifies the time unit is minutes.

h — Specifies the time unit is hours.

d — Specifies the time unit is days.

- **-memory [b | kb | mb | gb]**

A switch that specifies to return the memory usage. By default, this command returns the usage in bytes, but you can optionally specify one of the other units.

b — Specifies the memory usage unit is bytes.

kb — Specifies the memory usage unit is kilobytes.

mb — Specifies the memory usage unit is megabytes.

gb — Specifies the memory usage unit is gigabytes.

- **-hostname**

A switch that specifies to return the hostname.

- **-interactive**

A switch that specifies to return a “1” when the tool is in interactive mode or a “0” when executing from a dofile in batch mode.

Examples

Example 1

The following examples return information about the current tool:

get_tool_info -name

tessent_shell

get_tool_info -version

2012.4

get_tool_info -cpu_time

2.773

get_tool_info -memory

228343808.0

get_tool_info -hostname

galactica

Example 2

The following shows how to format the returned double value for the -memory switch:

**puts [format "Used memory %.2f GB." [get_tool_info -memory GB]]
//which prints the following:**

...

Used memory 1.26 GB.

The -cpu_time switch also returns double values.

Related Topics

[report_licenses](#)

[report_resources](#)

[set_tcl_shell_options](#)

[set_tool_options](#)

[get_tool_option](#)

Context: all contexts

Mode: all modes

Returns certain Tesson Shell options from the [set_tool_options](#) command.

Usage

```
get_tool_option -reapply_settings_after_reelaboration | -allow_vhdl_2008
```

Description

Returns certain Tesson Shell options from the [set_tool_options](#) command.

Arguments

- `-reapply_settings_after_reelaboration`

A switch that returns the re-elaboration settings for the loaded design.

- `-allow_vhdl_2008`

A switch that returns the whether any blocks contain VHDL 2008 constructs.

get_trace_flat_model_option

Context: all contexts

Mode: setup, analysis

Introspects the default behavior of the trace_flat_model command.

Usage

```
get_trace_flat_model_option [-controllability] [-latch] [-tag_condition] [-stop_condition]
                            [-stop_at_first_tag] [-max_levels] [-map_tag_to_design_module_boundary]
```

Description

Introspects the default behavior of the [trace_flat_model](#) command.

You can only specify one option at a time. The tool generates an error if more than one option is specified in a single command execution.

Arguments

- **-controllability**

An optional switch that returns the current setting of the controllability option. Possible return values are:

controlling — Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion.

constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected — With this value, gates are always traversed when reached.

- **-latch**

An optional switch that returns the current setting of the latch option. Possible return values are:

normal — A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of “-controllability connected” all latch input pins - including the latch enable pin - are traced through.

transparent — All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- **-tag_condition**

An optional switch that returns the current setting of the tag condition option. The returned value is an attribute expression. For more details about filtering attribute equation formats, refer to [“Attribute Filtering Equation Syntax”](#) on page 3162.

- **-stop_condition**

An optional switch that returns the stop condition for the trace. The returned value is an attribute expression.

- **-stop_at_first_tag**

An optional switch that returns the current setting of the -stop_at_first_tag option. This option specifies whether to interrupt tracing when the first tag object with the specified controllability is found. Possible return values are: {ON | OFF}

- **max_levels**

An optional switch that returns the current setting of the max_levels option. This option specifies the maximum level of gates to trace. Possible return values are: unlimited or a number.

- **-map_tag_to_design_module_boundary**

An optional switch that returns the current setting of the map_tag_to_design_module_boundary option, which specifies whether to map the tag points to gate_pins outside of library cells. Possible return values are: {ON | OFF}

Examples

Example 1

The following example returns the current setting for the -controllability option.

```
get_trace_flat_model_option -controllability
```

Example 2

The following example returns the current setting for the -tag_condition option.

```
get_trace_flat_model_option -tag_condition
```

Example 3

The following example returns the current setting for -stop_condition option.

```
get_trace_flat_model_option -stop_condition
```

Example 4

The following example returns the current setting for the -stop_at_first_tag option.

```
get_trace_flat_model_option -stop_at_first_tag
```

Example 5

The following example returns the current setting for the -max_levels option.

```
get_trace_flat_model_option -max_levels
```

Example 6

The following example returns the current setting for the -latch option.

```
get_trace_flat_model_option -latch
```

Related Topics

[set_trace_flat_model_options](#)

[trace_flat_model](#)

get_transcript_style

Context: unspecified, all contexts

Mode: all modes

Returns the keyword that specifies the style in which the tool transcripts commands.

Usage

```
get_transcript_style [-result_collection_limit | -argument_collection_limit |  
-one_collection_entry_per_line]
```

Description

Returns the keyword that specifies the style in which the tool transcripts commands.

The get_transcript_style command returns one of four keywords previously set with the [set_transcript_style](#) command. Possible return values are: off, full, tool_commands_only, and input_only.

Note that issuing the [set_transcript_style](#) command without an argument prints the style. Issuing get_transcript_style returns a value that you can assign to a variable for later use (see example below).

Arguments

- `-result_collection_limit`
An optional switch that specifies that the command is to return the number of collection entries shown in the interactive mode.
- `-argument_collection_limit`
An optional switch that specifies that the command is to return the number of collection entries shown in the listing when a collection is passed to a command as an argument.
- `-one_collection_entry_per_line`
An optional switch that specifies that the command is to return the collections of arguments and results with one entry per line.

Examples

The following example stores the current transcript style in a variable, turns off transcripting, then later restores the original transcript style:

```
set old_transcript_style [get_transcript_style]  
set_transcript_style off  
  
... # do something with transcripting turned off  
set_transcript_style $old_transcript_style
```

Related Topics

[echo](#)
[no_transcript](#)
[set_logfile_handling](#)
[set_transcript_style](#)

get_tsdb_info

Context: dft, patterns

Mode: setup, analysis, insertion

An introspection command used to extract information stored in the *.tsdb_info* files found in the *dft_inserted_design* directories of the TSDB directories.

Usage

```
get_tsdb_info [module_name] [-design_identifier id] [-silent]
  {-dictionary [-specified_module_only] | -exists | -level | -tsdb_info_file
  | -most_recent_design_identifier | -library_name | -ids_with_dates | -opened_tsdb_list
  | -child_blocks_dict | -version | -gate_extension }
```

Description

An introspection command used to extract information stored in the *.tsdb_info* files found in the *dft_inserted_designs* directories of the TSDB directory. Except for the -dictionary switch, all other switches apply to a specific *module_name* and *design_id* set. If the design is not elaborated in the session, you need to specify the *module_name* and potentially the *design_id* for which you want the requested information.

Arguments

- *module_name*

A string value that specifies the name of a module, or a collection containing a single module that identifies the module for which the *tsdb_info* is requested. If unspecified, then the tool uses the *module_name* returned by the [get_current_design](#) command.

- *-design_identifier id*

A switch and string pair that specifies the *design_id* associated with the design for which the *tsdb_info* is requested. You only need to specify this switch if the TSDB contains *dft_inserted_designs* directories associated with more than one *design_id*. The tool issues an error if there is more than one *design_id* available and this switch is not specified.

- *-silent*

A switch that suppresses the error message generated when a *module_name* and *design_id* combination is specified, and there is no *dft_inserted_designs* directory found for the combination in any of the opened TSDB directories. In this case, the tool returns a null value.

- *-dictionary*

A switch that returns the complete information set found in all *tsdb_info* files in the opened TSDBs. When this switch is specified, the tool ignores the specified *module_name* and

design_id values, and returns the information for all design and *design_id* pairs that exist. The format of the dictionary is as follows:

```
module_name {
    design_identifier {
        tsdb_info_file      <leaf name of .tsdb_info file>
        tsdb_info_version   integer
        creation_date       integer //epoch time
        level               chip | physical_block | sub_block
        icl_extraction_needed On | Off
        full_path           <relative path of .tsdb_info
                            file from current directory>
        library_name         <logical library name
    of module_name
        gate_extension       <extension of concatenated netlist>
        opened_tsdb_list     <list of opened tsdb(s) when created>
        child_blocks_dict {
            module_name design_identifier
            ...
        }
    }
}
```

- **-specified_module_only**

An optional switch used in conjunction with the -dictionary switch that returns only the dictionary for the specified *design_name/design_id*.

- **-exists**

A switch that returns a value that indicates whether a *design_name/design_id* set exists. Return values are as follows:

1 —Indicates the *design_name/design_id* set exists.

0 —Indicates the *design_name/design_id* set does not exist.

- **-level**

A switch that returns the design level associated with the specified *design_name/design_id* set. The returned value reflects the [set_design_level](#) command's value used when the tool created the *dft_inserted_design* directory.

- **-tsdb_info_file**

A switch that returns the full path name of the *.tsdb_info* file associated with the specified *design_name/design_id* set.

- **-most_recent_design_identifier**

A switch that returns the most recent *design_id* for the specified *design_name*.

- **-library_name**

A switch that returns the logical library name the specified *design_name* is associated to.

- **-ids_with_dates**

A switch that returns the list of *design_ids* associated with the specified *design_name*. The returned value is a list of list containing the following three elements: the *design_id* string; the date represented as an integer (epoch time); and the date represented in your local time zone.

- **-opened_tsdb_list**

A switch that returns the list of TSDB directories that were open when the tool created the *.tsdb_info* file. The paths are reported relative to the current working directory in Tesson Shell, unlike those in the file itself which are relative to the directory in which the tsdb info file resides. The first path in the list is always the relative path to the tsdb in which the tsdb info file resides.

- **-child_blocks_dict**

A switch that returns a dictionary listing the child blocks found immediately below the specified *design_name/design_id* set. The keys are the child block names, and the values are the *design_ids*.

- **-version**

A switch that returns the version of the *.tsdb_info* file. Return values are as follows:

1 — Indicates the current version.

0 — Indicates that there is no child block dictionary. If you specify the **-child_block_dict** switch, then the tool returns an empty dictionary.

- **-gate_extension**

A switch that returns the extension used by the concatenated netlist for the specified *design_name/design_id* set.

Examples

The following example illustrates the use of several switches of the **get_tsdb_info** command. Notice how the **format_dictionary** command is used to format the returned dictionary when the **-dictionary** switch is used.

```
help get_tsdb_info
get_tsdb_info [ <module_spec> ] [ -design_identifier <id> ] [ -silent ]
{ -dictionary | -exists | -level | -tsdb_info_file |
  -most_recent_design_identifier | -library_name | -ids_with_dates |
  -opened_tsdb_list | -child_blocks_dict | -version | -gate_extension }

puts [format_dictionary [get_tsdb_info -dictionary]]
```

```

chip {
  pass2 {
    tsdb_info_file chip.tsdb_info
    tsdb_info_version 1
    creation_date 1439219813
    level chip
    icl_extraction_needed On
    full_path simple_tsdb/dft_inserted_designs/chip_pass2.dft_inserted_design
    library_name work
    gate_extension vg
    opened_tsdb_list ./simple_tsdb
    child_blocks_dict {
      Pb1 pass2
      Sb1 pass2
    }
  }
  pass1 {
    tsdb_info_file chip.tsdb_info
    tsdb_info_version 1
    creation_date 1439219765
    level chip
    icl_extraction_needed On
    full_path simple_tsdb/dft_inserted_designs/
    chip_pass1.dft_inserted_design
    library_name work
    gate_extension v_gate
    opened_tsdb_list ./simple_tsdb
    child_blocks_dict {
      Pb1 pass1
      Sb1 pass1
    }
  }
}
Pb1 {
  pass2 {
    tsdb_info_file Pb1.tsdb_info
    tsdb_info_version 1
    creation_date 1439219800
    level physical_block
    icl_extraction_needed On
    full_path simple_tsdb/dft_inserted_designs/
    Pb1_pass2.dft_inserted_design
    library_name work
    gate_extension vg
    opened_tsdb_list ./simple_tsdb
    child_blocks_dict {}
  }
  pass1 {
    tsdb_info_file Pb1.tsdb_info
    tsdb_info_version 1
    creation_date 1439219759
    level physical_block
    icl_extraction_needed On
    full_path simple_tsdb/dft_inserted_designs/
    Pb1_pass1.dft_inserted_design
    library_name work
    gate_extension v_gate
    opened_tsdb_list ./simple_tsdb
  }
}

```

```

        child_blocks_dict {}
    }
}
Sb1 {
    pass2 {
        tsdb_info_file Sb1.tsdb_info
        tsdb_info_version 1
        creation_date 1439219783
        level sub_block
        icl_extraction_needed On
        full_path simple_tsdb/dft_inserted_designs/Pb1_pass2.dft_inserted_design
        library_name work
        gate_extension vg
        opened_tsdb_list ./simple_tsdb
        child_blocks_dict {}
    }
    pass1 {
        tsdb_info_file Sb1.tsdb_info
        tsdb_info_version 1
        creation_date 1439219754
        level sub_block
        icl_extraction_needed On
        full_path /simple_tsdb/dft_inserted_designs/Pb1_pass1.dft_inserted_design
        library_name work
        gate_extension v_gate
        opened_tsdb_list ./simple_tsdb
        child_blocks_dict {}
    }
}

```

puts [get_tsdb_info Sb1 -most_recent_design_id]

pass2

puts [get_tsdb_info Sb1 -design_id pass2 -level]

sub_block

puts [get_tsdb_info Sb1 -design_id pass2 -library_name]

work

puts [get_tsdb_info Sb1 -design_id pass2 -open_tsdb_list]

./simple_tsdb

puts [get_tsdb_info Sb1 -design_id pass2 -gate_extension]

vg

puts [get_tsdb_info Sb1 -design_id pass2 -version]

1

puts [get_tsdb_info Sb1 -design_id pass2 -tsdb_info_file]

./simple_tsdb/dft_inserted_designs/Sb1_pass2.dft_inserted_design/
Sb1.tsdb_info

puts [get_tsdb_info Sb1 -design_id pass2 -exists]

1

get_tsdb_list

Context: all contexts

Mode: all modes

Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

Usage

```
get_tsdb_list
```

Description

Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

When the default or specified TSDB output directory is already created, it is returned in the list even if there was no explicit open_tsdb command for it.

Arguments

None

Examples

The following example shows the use of the get_tsdb_list command before and after some are closed.

```
get_tsdb_list  
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.tsdb  
/home/projects/chipa/tsdbs/corec.tsdb  
  
close_tsdb [list ${tsdb_home}/corea.tsdb ${tsdb_home}/coreb.tsdb ]  
  
get_tsdb_list  
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

[close_tsdb](#)

[set_tsdb_output_directory](#)

get_tsdb_output_directory

Context: unspecified, dft, patterns

Mode: setup, analysis, insertion

Returns the directory used by the process_dft_specification command which is the directory that has been previously set by the set_tsdb_output_directory command.

Usage

```
get_tsdb_output_directory
```

Description

Returns the directory used by the [process_dft_specification](#) command which is the directory that has been previously set by the [set_tsdb_output_directory](#) command.

The TSDB output directory defaults to *./tsdb_outdir* until the set_tsdb_output_directory command has been invoked. When either the get_tsdb_output_directory or set_tsdb_output_directory command is invoked, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it does not.

The process_dft_specification command is currently the only command that makes use of the TSDB directory.

Arguments

None

Examples

Example 1

The following example stores the TSDB output directory into a Tcl variable called tsdb_outdir.

```
set tsdb_outdir [get_tsdb_output_directory]
```

Example 2

The following example shows a possible error that can be generated when invoking the command.

```
set_tsdb_output_directory dir1
exec "rm -r dir1; touch dir1"
get_tsdb_output_directory

// Error: The specified tsdb_output_directory 'dir1' exists but is not a
// directory.
// Error: Directory was either deleted or its write permission changed
// between now and the invocation of set_tsdb_output_directory command.
```

Related Topics

[set_tsdb_output_directory](#)

get_validated_objects

Context: dft, patterns

Mode: setup, analysis, insertion

Validates the values supplied to arguments or switches of a command created with the register_tcl_command command.

Usage

```
get_validated_objects object_spec [-allowed_object_type_list allowed_object_type_list]
    | [-argument_name argument_name] | -switch_name switch_name]
```

Description

This command accepts a list of object names or a collection of objects, and that verifies if their types are present in the allowed object type list. When this condition is met, the tool returns a collection with all object names converted to objects.

If you are validating the value of a switch, specify this value using the -switch_name switch. For example, if you are validating the value provided to a switch called -below_instance, specify “-switch_name below_instance”. The error messages will start with “Supplied -below_instance value contains...”

If you are validating a positional argument, use the -argument_name switch to specify the argument’s name. For example, if you are validating the <port_spec> argument, then specify -argument_name port_spec. The error messages will start with “Supplied <port_spec> contains...” If you do not specify either the -argument_name switch or the -switch_name switches, the error messages will start with “Supplied <node_spec> contains...”.

Arguments

- ***object_spec***

A Tcl list with one or more object names or a collection containing one or more objects.

- **-allowed_object_type_list *allowed_object_type_list***

A switch-value pair that specifies which object types are allowed. The value is a Tcl list of one or more object names. The allowed object names are as follows:

- port
- pseudo_port
- pin
- net
- module
- instance

- gate_pin
- icl_port
- icl_pin
- icl_module
- icl_instance

When unspecified, the default allowed object type list is {port pin net pseudo_port gate_pin}.

- **-argument_name *argument_name***

A switch value pair that specifies the name of the argument to be validated. Specifying this option affects the error messages that are generated when the object_spec contains invalid objects. When both the -argument_name and the -switch_name switches are not specified, the error messages start with “Supplied <node_spec> contains...” When the -argument_name switch specifies a value xxx, then the error messages start with “Supplied <xxx> contains...”

For example, if you are validating the <port_spec> argument, then specify -argument_name port_spec, and the error messages will start with “Supplied <port_spec> contains...”

- **-switch_name *switch_name***

A switch value pair that specifies the name of a switch value to be validated. Specifying this option affects the error messages that are generated when the object_spec contains invalid objects. When both the -argument_name and the -switch_name switches are not specified, then the error messages start with “Supplied <xxx> contains...” When the -switch_name switch specifies a value xxx, then the error messages start with “Supplied -xxx value contains...”

For example, if you are validating the value provided to a switch called -below_instance, specify -switch_name below_instance, and the error messages will start with “Supplied -below_instance value contains...”

Examples

The following example uses the get_validated_objects command with the body of a registered Tcl command to validate the argument of a command having this syntax:

```
get_instance_leaf_name instance_spec
```

The proc implementing this command uses the `get_validated_objects` as follow:

```
proc get_instance_leaf_name.body {args} {
    array set ARGS {instance_spec ""}
    array set ARGS $args

    set instance_objects [get_validated_objects $ARGS(instance_spec) \
        -argument_name instance_spec \
        -allowed_object_type_list instance]

    #If I make it hear, instance_objects contains a collections of instances
}
```

get_write_patterns_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis, setup

Returns introspection results from either port lists or vector callbacks.

Usage

Usage for Port Lists

```
get_write_patterns_options -existing_used_ports | -additional_port_list | -all_used_port_list  
| -port_direction port_direction | -port_offstate port_offstate
```

Usage for Vector Callbacks

```
get_write_patterns_options -vector_callback_list [-trigger {before_first | after_last |  
every_cycle}] | -received_callback_port_list received_callback_port_list  
| -returned_callback_port_list returned_callback_port_list
```

Description

Returns introspection results from either port lists or vector callbacks.

Arguments

- **-existing_used_ports**

A required switch that returns the current subset of existing design ports as a Tcl list of port names or gate_ids. The list is in the order that the ports are listed in the final pattern file, minus the additional ports. If there is no subset of existing_used_ports active, the full list of default design ports is returned.

- **-additional_port_list**

A required switch that returns a Tcl list of lists of the additional port names, and port directions. This is the order that the additional ports will be listed in the final pattern file that will be written. If there is no additional ports specified, an empty list is returned. The format of the list of lists is as follows:

```
{ { port_name direction} { port_name direction} ...}
```

- **-all_used_port_list**

A required switch that returns the current subset of existing design ports and any specified additional ports as a Tcl list of lists of the port names and port directions. If no subset of existing design ports is active, then the default list of design ports plus any additional ports are returned. The format of the Tcl list is the same as that used for -additional_port_list.

- **-port_direction *port_direction***

A required switch and string pair that returns the port direction for a named port, where the port direction is one of input, output, or inout. This command works on all ports, including any additional ports added with -additional_port.

- **-port_offstate *port_offstate***

A required switch and value pair that returns the binary offstate state value of ‘0’ or ‘1’ for a named port if the port is a clock port. If the port is not a clock port, then the state value of ‘N’ is returned.

- **-vector_callback_list [-trigger {before_first | after_last | every_cycle}]**

A required switch and literal pair that returns the current vector callback proc names as a Tcl list. If there are no callbacks, an empty list is returned. If the –trigger option is used, only the name of the callback for that trigger is returned.

- **-received_callback_port_list *received_callback_port_list***

A required switch and value pair that returns the callback ports as a TCL list of names in the same order as the port_values will be present in the vector_set. This introspection command can be used within vector_callback procs in order to have the list of ports and the order that will be present in all port_values strings within vectors.

- **-returned_callback_port_list *returned_callback_port_list***

A required switch and value pair that returns the callback ports as a TCL list of names in the same order as the port_values will be present in the vector_set. This introspection command can be used within vector_callback procs in order to have the list of ports and the order that will be present in all port_values strings within vectors.

Examples

Example 1

This example show the results of this command if called after the port_list has been set.

```
ANALYSIS> puts [join [get_name_list [get_write_pattern_options -all_used_port_list]] "\n"]
ijtag_si input
ijtag_so output
ijtag_tms input
ijtag_tck input
myNewInput input
myNewOutput [1] output
myNewOutput [0] output
```

Related Topics

[report_write_patterns_options](#)

[set_write_patterns_options](#)

[write_patterns](#)

[Vector Creation and Modification](#)

get_xclock_handling

Context: all contexts

Mode: all modes

Returns the options specified with the [set_xclock_handling](#) command.

Usage

```
get_xclock_handling [ -pessimistic_simulation ]
```

Description

Returns the options specified with the [set_xclock_handling](#) command, which specifies whether the sequential element model outputs X when any of its clock inputs become X.

Arguments

- `-pessimistic_simulation`

An optional switch that returns the setting (“1” (on) or “0” (off)) that affects the output of the edge-triggered sequential element model when the clock port becomes X as shown in [Table 6-22](#) on page 2465.

Related Topics

[set_xclock_handling](#)

help

Context: unspecified, all contexts

Mode: all modes

Lists commands and associated usage syntax.

Usage

`help [command_name] [-manual] [-all]`

Description

Lists commands and associated usage syntax.

Specifying the help command without any arguments returns a list of all available commands. Specifying a single command name displays its usage syntax.

Arguments

- *command_name*

An optional string that specifies the name of the command for which you want help. Entering the first part of a command name returns a list of commands that start with the specified string. The *command_name* can include wildcard characters. For example, you can use an “*” to match zero or more characters at the position of the asterisk in the string.

You also can use the tab completion method on available commands, which supplies missing characters between underscores. For example, specifying “help se_sy_m” and pressing the Tab key inserts “help set_system_mode” on the command line. Tab completion also works in this same way when invoking any tool command.

If a single command is returned, its usage syntax also displays.

- *-manual*

An optional switch that displays the reference manual description for the specified command. If you use this switch without specifying a command name, the tool opens the Tesson documentation system, giving you access to all the manuals for the product group.

- *-all*

An optional switch that displays all command matches, including unavailable commands, which are identified with the “(...unavailable...)” notation. Specifying “help *unavailable_command_name*” displays the reason why a particular command is unavailable.

Examples

Example 1

The following example returns the usage syntax for the history command:

help history

```
history [ <count> ] [ -Nonumbers ] [ -Reverse ] [ -Save <savefile> ]
```

Example 2

The following example returns all commands in the current mode that begin with the string *set_design*:

```
help set_design
```

```
set_design_include_directories  
set_design_macros  
set_design_sources
```

Example 3

The following example returns all commands in the current mode that contain the string *design*:

```
help *design*
```

```
delete_design  
get_current_design  
set_current_design  
set_design_include_directories  
set_design_macros  
set_design_sources  
write_design
```

Example 4

The following example returns all commands containing the string *design*, including those that are unavailable:

```
help *design* -all
```

```
delete_design  
find_design_names      (...unavailable...)  
get_current_design  
set_current_design  
set_design_include_directories  
set_design_macros  
set_design_sources  
write_design
```

history

Context: unspecified, all contexts

Mode: all modes

Displays a list of previously executed commands.

Usage

`history [list_count] [-Nonumbers] [-Reverse] [-Save filename]`

Description

Displays a list of previously executed commands.

You can use command line editing to recall a previous command and edit it for submission as a new command.

A leading number precedes each command line in the history list that indicates the order in which the commands were entered.

Arguments

- *list_count*
An optional integer that specifies for the tool to display only the specified number (*list_count*) of most recent executed commands. If no *list_count* is specified, the tool displays all previously executed commands.
- -Nonumbers
An optional string that specifies for the tool to display the history list without the leading numbers. This is useful for creating dofiles. The default displays the leading numbers.
- -Reverse
An optional switch that specifies for the tool to display the history list starting with the most recent command rather than the oldest.
- -Save *filename*
An optional switch and string pair that specifies the name of a file to which to write the command history. This is useful for creating dofiles.

Examples

Example 1

The following command saves the command history into a dofile for later execution.

history -save dofile1.txt

Example 2

The following command displays the history list with leading numbers, starting with the oldest command.

history

```
1 help hist
2 dof instructor/fault.do
3 set_system_mode analysis
4 set_fault_type stuck
5 add_faults -all
6 create_patterns
7 report_statistics
8 report_faults -class ATPG_UNTESTABLE
9 analyze_fault /I$20/en -stuck_at 1
10 set_system_mode setup
11 set_system_mode analysis
12 set_fault_type iddq
13 add_faults -all
14 create_patterns
15 report_statistics
16 history
```

iApply

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Triggers the retargeting of all queued iRead and iWrite commands.

Usage

`iApply [-end_in_pause]`

Description

Triggers the retargeting of all queued iRead and iWrite commands.

This command causes the retargeting engine to calculate the optimal sequence of scan loads, primary input stimulations, and primary output observations to satisfy all queued iRead targets and iWrite targets, as illustrated in [Figure 4-5](#).

The number of ParallelIO time frames and/or Scan time frames used to satisfy all queued up iWrite and iRead commands depends on the number of queued commands and the circuit topology described in the ICL description.

The solution criteria for the iWrite commands is that all specified iWrite values are achieved and present at the end of the iApply time frame. The order in which they become satisfied is unknown and is chosen by the solver in order to minimize the number of clock cycles needed to complete the tasks. The solver is designed to minimize transition on the writable objects and will limit them to a single transition per iApply time frame if the ICL circuit allows it.

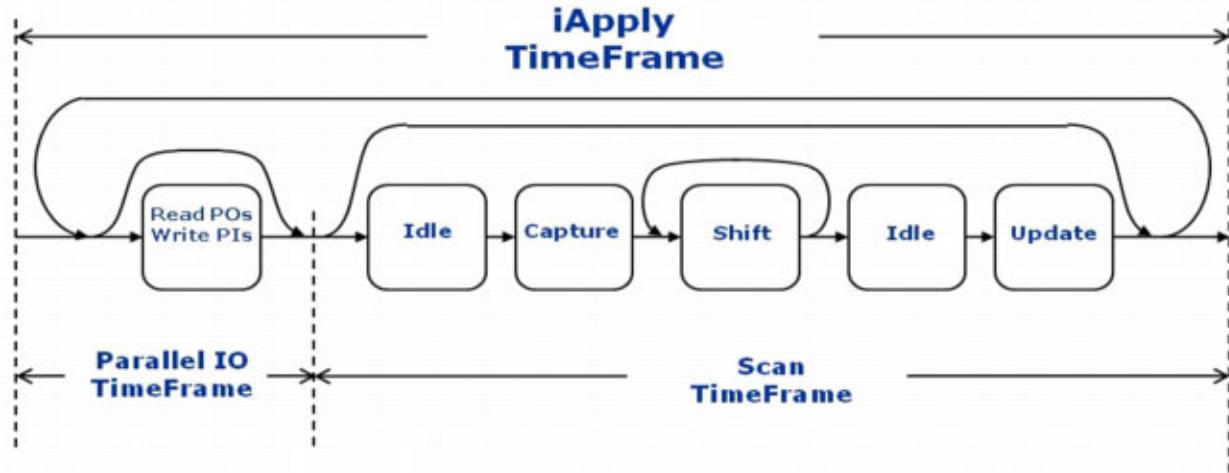
If it is not possible to achieve all specified iWrite values at the end of the time frame, the specified iWrite command set is said to have no solution. The iRead object only needs to be observed once anywhere within the time frame. The value is strobed at the first opportunity that it becomes observable and is then ignored if re-observed on subsequent Scan or parallel IO time frames.

If there is no solution for the retargeting task, the iRead targets and iWrite targets are taken out of the queue and the current state of the circuit is retained. An error message appears in this case. If there is a solution, the solution is appended to the opened pattern set. The circuit state is updated to reflect the situation after the actions within the current iApply time frame have been applied.

You can use the optional `-end_in_pause` switch to help create IJTAG scan chain integrity tests. This switch allows you to observe the same values at the scan output that have been shifted in

before. When you do not use this switch, the shift register content is always overwritten by the capture values before being observed at the scan output.

Figure 4-5. iApply Time Frame



Arguments

- `-end_in_pause`

An optional switch that helps you to create JTAG scan chain integrity tests. The `-end_in_pause` switch changes the handling of the last scan load of the iApply and the first scan load of the next iApply.

Note that the following explanation refers to states of a finite state machine (FSM) of an IEEE 1149.1 (JTAG) TAP controller. “IR” is instruction register, and “DR” is data register. For a complete explanation about how the TAP controller operates, refer to the IEEE 1149.1 standard document.

When you use the `-end_in_pause` switch, and the last scan load is a “TAP scan load” (capture, shift and update activity is controlled by a TAP state machine), then the TAP state machine ends in the pause-DR or pause-IR state (that is, the tool performs no update-DR/update-IR). The first scan load of the next iApply starts from there and performs another scan load (obviously without capture-DR/capture-IR) before returning to RTI (run test/idle).

When you use the `-end_in_pause` switch, and the last scan load is an ordinary scan load (capture, shift, and update activity is controlled by the appropriate CaptureEnPort, ShiftEnPort, UpdateEnPort), then the “update” of the last scan load of the iApply is skipped. The first scan load of the next iApply skips the “capture.” For more information about the CaptureEnPort, ShiftEnPort, UpdateEnPort keywords, refer to “ICL and PDL Modeling” in the [Tessent IJTAG User’s Manual](#).

You can use this switch only when the iWrite targets are all part of ScanRegisters. Those ScanRegisters must be configurable to appear as one concatenated scan chain.

Return Values

None

Examples

Example 1

In the following example, two iWrite and two iRead commands are queued up when the iApply command is issued. The solver starts from the current state of the ICL circuit and looks for a series of ParallelIO and Scan time frames to have u1.ina equal to “11” and u1.inb equal to “0” at the end of the iApply time frame, and to have observed at least once the u1.out1 and u1.out2 objects.

The number of ParallelIO and Scan time frames needed to achieve the objective depends on the ICL network. If both u1.ina and u1.inb shared a common source in ICL, then the given PDL could not be satisfied because it would be impossible to have u1.in1 high with u1.inb low together at the end of the iApply time frame.

```
iWrite u1.ina 0b11  
iWrite u1.inb 0b0  
iRead u1.out1  
iRead u1.out2 0x7  
iApply
```

Example 2

The following example applies a chain integrity test.

```
open_pattern_set chain_pattern  
    iWrite myRegister 0b00110011  
    iApply -end_in_pause  
    iRead myRegister 0b00110011  
    iApply  
close_pattern_set
```

Related Topics

[iRead](#)
[iWrite](#)

iCall

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tesson Shell](#)” in the *Tesson IJTAG User's Manual*.

Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.

Usage

`iCall [icl_instance_path.]iproc_name [arguments...]`

Description

Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.

The *effective_icl_instance_path* is computed by concatenating the following three components:

- iProc instance path — Path that exists when running commands specified within an iProc; it is set to the *effective_icl_instance_path* used in the iCall command that called the iProc. This value is empty when the iCall is not specified within an iProc.
- iPrefix path — Path specified as the argument to the iPrefix command since the beginning of the [open_pattern_set](#) for commands that are not inside an iProc, or since the beginning of the iProc for commands that are inside an iProc. If no iPrefix command has yet been specified in the current scope, this value is empty.
- *icl_instance_path* — An optional path specified in front of the *iproc_name* inside the iCall command, or in front of the *icl_object* inside other PDL commands such as [iWrite](#), [iRead](#), and [iClock](#).

The *effective_icl_instance_path* is illustrated in the examples that follow.

The other arguments of this command are passed to the iProc in the same order as they appear in the command line. You must specify at least as many arguments as there are variables without default values in the corresponding iProc specification. For arguments defined with a default value in the associated iProc, their value takes on the default value when omitted.

Error Reporting

Similar to ordinary Tcl procs, iProcs can be nested, and this nesting can even be a mixture of iProcs and ordinary Tcl procs. An iProc can be called from within an ordinary Tcl proc which in turn has been called from within another iProc. If an error occurs in such a situation, it is often desirable to know the exact call stack to find the root-cause of the error. Tesson Shell reuses the

Tcl infrastructure to provide this sort of information. The built-in variable “errorInfo” contains useful data about the call stack and the location where the erroneous command can be found.

In case of an error, the Tcl interpreter of Tessent Shell fills the variable appropriately, but it treats iProcs and ordinary Tcl procs in the same way. Therefore, the content of the “errorInfo” variable shows the internal representation of the failing iProc. This internal representation consists of one or more namespace identifiers, the name of the module for which the iProc is registered and the name of the iProc. These parts are separated by double colons.

Example

```
ANALYSIS> puts $errorInfo
can't read "iDoNotExist": no such variable
    while executing
        "puts $iDoNotExist"
            (procedure "::mentor::iprocs::module1::run_test" line 21)
                invoked from within
                    "::mentor::iprocs::module1::run_test {blue}"
                        invoked from within
                            "iCall block2_I1.inst1.run_test blue"
```

The content of the variable “errorInfo” shows a procedure called “::mentor::iprocs::module1::run_test”. This refers to the iProc “run_test”, which is registered for the module “module1” in the iProc namespace “::mentor::iprocs”. The lines above also show the command by which this iProc has been called (“iCall block2_I1.inst1.run_test blue”) and the root cause of the error (can't read “iDoNotExist”: no such variable).

Arguments

- **[icl_instance_path.]iproc_name**
Name of an iProc preceded by an optional *icl_instance_path* and separated by a dot. The specified iProc must have been registered against the ICL module associated with the *effective_icl_instance_path*.
- arguments...
Additional optional arguments that are passed to the iProc procedure. The number of required arguments depends on the definition of the iProc and on the existence of default values for the iProc arguments.

Return Values

None. Unlike normal Tcl proc, even if the iProc contains the Tcl return command, the specified value is not returned by the iCall command which is consistent with the IEEE 1687 standard.

Examples

The following example assumes an ICL module called myInstrument1 exists and this module is instantiated four times below the current design with the following instance paths:

```
core1_I1.myInst1_I1core1_I1.myInst1_I2core1_I2.myInst1_I1core1_I2.myInst1_I2
```

The ICL module myInstrument1 has an ICL sub-module called myInstrument1_decode_logic which is instantiated within myInstrument1 as “decode_logic”.

This numbered example illustrates the declarations of iProcs against the myInstrument1 and myInstrument_decode_logic modules, and the use of the iCall command to execute them against various instances of myInstrument1.

Line 1 issues the first iProcsForModule command for myInstrument1 which means iProcs declared afterward are registered against that module. For that reason, the iProcs declared at line 2 and 12 are registered against myInstrument1.

Line 24 issues the second iProcsForModule command which means the next iProcs are registered against the module myInstrument1_decode_logic.

The commands starting at line 41 are not inside an iProc so the **iProc instance path** value is empty when computing the effective_icl_instance_path for those commands.

The iCall on line 42 has an effective_icl_instance_path of core1_i1.myInst1_i1 because **iPrefix path** is currently set to core1_i1 (from the **iPrefix** command on line 41) and **icl_instance_path** is specified as myInst1_i1.

The iCall on line 42 specifies the value for the cnt argument while the second argument is left to the default value specified within the iProc declaration.

The iCall on line 43 overwrites the value of the second argument. Note that the first argument is required because it has no default values defined in the iProc declaration.

The iCall on line 5 is inside of an iProc, so it has an **iProc instance path** value that is equal to the effective_icl_instance_path value of the iCall that called the iProc. Looking at the iCall command on line 5 when called by the iCall on line 41, its effective_icl_instance_path is core1_i1.myInst1_i1.decode_logic because **iProc instance path** is core1_i1.myInst1_i1, **iPrefix path** is decode_logic, and **icl_instance_path** is empty.

The ICL instance core1_i1.myInst1_i1.decode_logic is an instance of ICL module myInstrument1_decode_logic and the proc Setup is registered against this module on line 25 because it follows the iProcsForModule command specified on line 24.

```
1 iProcsForModule myInstrument1
2 iProc run_testa {cnt {mode red}} {
3     iCall CheckStart
4     iPrefix decode_logic
5     iCall Setup $cnt $mode
6     iCall run $mode
7     iPrefix
8     iCall decode_logic.Status
9     iRead decode_logic.cnt $cnt
10    }
11 }
12 iProc CheckStart {} {
13     iRead go Fail
14     iRead done No
15     iApply
16 }
17 iProc run {mode} {
18     set cycles(red) 50
19     set cycles(blue) 90
20     set cycles(green) 70
21
22     iRunLoop $cycles($mode) -tck
23 }
24 iProcsForModule myInstrument1_decode_logic
25 iProc Setup { cnt mode } {
26     iWrite mode $mode
27     if {$mode == "red"} {
28         iWrite p1 0b0
29         iWrite p1(2) 0b1
30     }
31     iApply
32     iWrite enable Yes
33     iApply
34 }
35 iProc Status {} {
36     iRead done Yes
37     iRead go Pass
38     iRead count $cnt
39     iApply
40 }
41
41 iPrefix core1_i1
42 iCall myInst1_i1.run_testa 5
43 iCall myInst1_i2.run_testa 10 blue
44 iPrefix
45 iCall core1_i2.myInst1_i1.run_testa 15 red
```

Related Topics

[ALL_MAX_LOOP](#)

[iPrefix](#)

[iProc](#)

[iProcsForModule](#)

[iWrite](#)

iClock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Checks whether a controlling clock path from a valid clock source to the specified clock port exists.

Usage

`iClock [icl_instance_path.]icl_clock_port`

Description

Checks whether a controlling clock path from a valid clock source to the specified clock port exists.

This command takes into account the [iClockOverride](#) specifications and the FreqMultiplier and FreqDivider properties in the ICL description.

Prior to issuing this command, clock path sensitization must have been performed using the appropriate iWrite and iApply commands such that SelectedBy sources of ICL ClockMultiplexers seen along the traced path have known constant values.

If clock tracing fails before reaching a primary input, an error is generated. If the primary input is reached but does not have an associated [add_clocks](#) command to specify the clock is active with a known period, an error is also generated.

The cumulative effective frequency multiplier is computed along the traced clock path and the final number is stored against the specified clock port. This value is later used to normalize the iRunLoop command to a common tester period as described in the [iRunLoop](#) command description.

Arguments

- `[icl_instance_path.]icl_clock_port`

Name of an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The [iCall](#) command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

Return Values

None

Examples

In the following example, an iClock command is first specified to check that the reference source pin of the PLL has a controlling source to a primary input of the current design, and that the port has a specified clock period as defined by an [add_clocks](#) command.

The iWrite and [iApply](#) commands then program the PLL to a multiplication factor of 1.5 and set the clock multiplexer to the desired state.

The result of the PLL setup is reflected by the [iClockOverride](#) command such that the iClock definition on inst1.mbist_i1.clk, which traces to the PLL, properly computes the correct multiplication factor.

The first iRunLoop command specifies that the PLL will lock in less than 200 reference clock cycles. The second iRunLoop results in a loop of 4000 (6000*2/3) clock cycles at the current design boundary. For more information on the iRunLoop scaling algorithm, see the iRunLoop command description.

```
iClock inst1 pll.ref
iWrite inst1 pll_setup.mult 0b0100
iWrite inst1 pll_setup.source 0b01
iApply
iClockOverride inst1 pll.vco -freqMultiplier 3 -freqDivider 2
iRunLoop 200 -sck inst1 pll.ref
iClock inst1 mbist_i1 clk
iRunLoop 6000 -sck inst1 mbist_i1 clk
```

Related Topics

[add_clocks](#)
[delete_clocks](#)
[get_iclock_list](#)
[get_iclock_option](#)
[iClockOverride](#)
[report_clocks](#)
[report_icl_modules](#)

iClockOverride

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Models how the functional clocking has been programmed.

Usage

```
iClockOverride [icl_instance_path.]icl_clock_port -source [icl_instance_path.]icl_clock_port  
[-freqMultiplier multiplier] [-freqDivider divider]
```

```
iClockOverride [icl_instance_path.]icl_clock_port -period period [s | ms | us | ns | ps | fs | as]
```

Description

Models how the functional clocking has been programmed.

System clocking syntax inside ICL is adequate to describe a constant frequency multiplication and division number. With ClockMultiplexers in ICL, it is possible to model alternative sources as long as the select lines of the multiplexers are controlled by ijtag registers. The functional clocking path of ijtag instruments is not required to be controlled by ijtag itself. As long as the current functional clocking is properly modeled, the [iRunLoop](#) commands can be normalized correctly.

You should have a way to set up the functional clock using the ijtag network if you need to change it between patterns. You can use the complete syntax available inside the test_setup procedure to program an arbitrary functional clock system. You simply use this iClockOverride command to model how the functional clocking was programmed. Even if the multiplication factor is controlled by ijtag registers, you use the iClockOverride command to describe the effect of the newly written value. This is illustrated in the [iClock](#) command example.

You can use the iClockOverride command to override only the source, or only the cumulative multiplication/divider value, or both. If the -source option is not specified, the source is unchanged; it remains the source documented in ICL, or the last iClockOverride -source value that may have been specified within the current open pattern set.

If the -freqDivider and the-freqMultiplier arguments are both specified, both are overridden. Specifying just -freqDivider without -freqMultiplier or vice versa, sets the unspecified option to 1. If neither of the -freqDivider and the-freqMultiplier arguments are specified, the previous -freqDivider and the-freqMultiplier values are preserved, regardless of whether they came from ICL or a previously specified iClockOverride.

Arguments

- **[icl_instance_path.]jicl_clock_port**

A required value that specifies an internal clock port defined inside the ICL description. The specified clock port must be a port declared as a ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The [iCall](#) command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- **-source [icl_instance_path.]jicl_clock_source_port**

An optional switch and value pair that specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The [iCall](#) command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- **-freqMultiplier multiplier**

An optional switch and value pair that overrides the frequency multiplication factor between the *[icl_instance_path.]jicl_clock_port* and its source. It's value defaults to 1 when - freqDivider is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]jicl_clock_port*

- **-freqDivider divider**

An optional switch and value pair that overrides the frequency ratio between the *[icl_instance_path.]jicl_clock_port* and its source. It's value defaults to 1 when - freqMultiplier is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]jicl_clock_port*

- **-period period**

An optional switch and value pair with optional time unit that overrides the period of the ToClockPort. The default time unit is “ns”. A ToClockPort with a specified period serves as an internal clock source, that is, as an oscillator.

Return Values

None

Examples

The following example assumes both inst1.inst2.sysclock and clk2 are declared as ToClockPorts in the ICL description. The first iClockOverride command only overrides the source and leaves the multiplication and division number unchanged. The second iClockOverride command overrides the frequency multiplication factor to 10 and the frequency division ratio to 1.

```
iClockOverride inst1.inst2.sysclock -source clk2
iClockOverride inst1.inst2.sysclock -freqMultiplier 10
```

Related Topics

[add_clocks](#)

[delete_clocks](#)

[get_iclock_list](#)

[get_iclock_option](#)

[iClock](#)

[report_clocks](#)

[report_icl_modules](#)

iComparePort

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iTOPProc that is called during the processing of a procfile. For more information, see [“IJTAG and ATPG in Tessent Shell”](#) in the *Tessent IJTAG User's Manual*.

Allows you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.

Usage

iComparePort *object_spec value* [-after *state_transition_list*]

Description

Allows you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.

An iComparePort or an iForcePort command is only allowed in an iTOPProc and not in a normal iProc as it is not retargetable upward.

It is best to keep the iProcs that only interact with the ijtag ports separated from the ones using the iComparePort command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iComparePort is applied on the next [iApply](#) command or the next [iRunLoop](#) command and lasts until it is iCompared to an X. You can use the -after switch to delay the compare and have it appear only after a specific state transition.

An iComparePort command remains active until one of the following events happen:

- The iComparePort is overwritten with another iComparePort command or — in case of bidirectional ports — with another iForcePort command.
- The pattern set is closed and the next pattern set is opened without the -no_initial_ireset switch. (Note: this only applies to the context “patterns -ijtag”).
- If an iComparePort command is part of an iTOPProc which is called from within an iMerge block, then the lifetime of the iComparePort command is limited to the retargeting of that iTOPProc.

This is done for the following reason: If there are conflicting iComparePort specifications in concurrent iCalls, the associated iTOPProcs must be retargeted in series. If the iComparePort specifications survived the end of the iTOPProc, the active

iComparePort after the iMerge –end would depend on the (undefined) order of the retargeting of the concurrent iCalls. This non-deterministic behavior must be avoided.

You can use this command in an iCall used as part of test_setup for ATPG, however, it makes the iTopProc un-retargetable upward. You should avoid doing so at least until you reach the chip level.

Arguments

- *object_spec*

A collection containing the top level output or bidirectional port/ports. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to compare. Use “read_verilog -interface_only” or “read_vhdl -interface_only” to read in the port list prior to issuing the set_current_design command as shown here:

```
set_context patterns -ijtag -no_rtl
read_verilog top.v -interface_only
read_icl top.icl
set_current_design top
```

- *value*

A string representing the value to compare ‘0’, ‘1’, ‘X’ or ’Z’. You can use the 0b prefix to specify a multi-bit binary number or the 0x prefix to specify an hexadecimal number. The X value means to stop comparing the port.

- -after *state_transition_list*

An optional switch-value pair used to specify a Tcl list of state transitions after which the compare is active. The compare is active in the cycle after one of the listed state transitions has happened and is off otherwise. The list of available state transitions are all the ones supported by the IEEE1149.1 state machine and state names are the same names used in SVF.

RESET, IDLE,

DRSELECT, DRCAPTURE, DRSHIFT, DREXIT1, DRPAUSE, DREXIT2, DRUPDATE
IRSELECT, IRCAPTURE, IRSWIFT, IREXIT1, IRPAUSE, IREXIT2, IRUPDATE

Examples

The following example compares two bits of a bussed port during the entire scan load needed to iWrite the TDR.

```
iComparePort GPIO[3:0] 0b1XX0
iWrite tdr[7:0] 0b0
iApply
```

The following example performs the same comparison as the previous example except that the port is compared when the state machine goes from DRSelect to DRCapture which corresponds to when the boundary scan cell captures the input ports.

```
iComparePort GPIO[3:0] 0b1XX0 -after DRSELECT:DRCAPTURE  
iWrite tdr[7:0] 0b0  
iApply
```

Related Topics

[iApply](#)
[iForcePort](#)
[iProc](#)
[ProcsForModule](#)

identify_redundant_faults

Context: dft -edt, patterns -scan

Mode: analysis

Identifies redundant faults among current faults whose redundancy has not yet been determined.

Usage

identify_redundant_faults

Description

Note

 By default, the [create_patterns](#) command automatically performs redundant fault identification at the conclusion of the run.

Identifies redundant faults among current faults whose redundancy has not yet been determined.

Identifies combinationally redundant (RE) faults from among those faults in the current fault list whose redundancy has not been determined yet.

This command is useful in situations where the tool has not specifically checked some faults for redundancy, but where establishing their redundancy might improve test coverage. The following are examples of such situations:

- When the tool generates patterns using named capture procedures, it does not identify RE faults. You might improve test coverage by using this command at the end of such an ATPG session.
- When the tool simulates external patterns, it does not identify RE faults among the undetected faults. You might improve test coverage by using this command after the simulation.

This command will only target faults not yet proven to be either RE or non-RE. For example, the command will not target a fault currently classified as det_simulation (DS) or that was classified as DS before but is now uncontrolled (UC) or unobserved (UO). The command is meaningful only when the current fault type is stuck-at, bridge, transition, or UDFM.

Arguments

None

Examples

Example 1

The following example creates patterns using named capture procedures, then identifies the RE faults:

```

set_system_mode analysis
set_capture_procedures on -all
create_patterns

...
// There may be unidentified redundant faults when generating patterns
// with named capture procedures defined, which may lead to lower test
// coverage being reported.
// To perform redundant fault identification, invoke the command:
//     identify_redundant_faults

...
Statistics Report
-----
Fault Classes          #faults    #faults
                      (coll.)    (total)
-----
FU (full)             471724    783223
-----
UC (uncontrolled)      39        173
UO (unobserved)       2748     4490
DS (det_simulation)   380299    652427
DI (det_implication)  66661     88877
PU (posdet_untestable) 26        36
PT (posdet_testable)  147       223
BL (blocked)          1907     3010
AU (atpg_untestable)  19897    33987
-----
Coverage
-----
test_coverage          95.17%   95.05%
fault_coverage         94.79%   94.68%
atpg_effectiveness    99.41%   99.40%
-----
#test_patterns          2736
#clock_sequential_patterns 2736
#simulated_patterns    2880
CPU_time (secs)        793.1
-----
identify_redundant_faults

```

```
// deterministic ATPG invoked with comb/seq abort limit = 0/2
// -----
// Performing redundant fault identification for 22857 faults
// -----
// # red.    # non-red.  # abort   # remn.   progress   test      process
// faults     faults    faults    faults      coverage    CPU time
// 261        12977     97       9619      57.92%    95.22%    3.26 sec
// 1361       16215     421      5281      76.90%    95.45%    4.80 sec
// 2846       18269     849      1742      92.38%    95.75%    6.67 sec
// 3121       18896     922          0      100.00%   95.81%    6.89 sec
//
```

report_statistics

Statistics Report

Fault Classes	#faults (coll.)	#faults (total)
FU (full)	471724	783223
UC (uncontrolled)	39	173
UO (unobserved)	2290	3832
DS (det_simulation)	380299	652427
DI (det_implication)	66661	88877
PU (posdet_untestable)	26	36
PT (posdet_testable)	145	221
BL (blocked)	1907	3010
RE (redundant)	3121	4322
AU (atpg_untestable)	17236	30325

Coverage		
test_coverage	95.81%	95.58%
fault_coverage	94.79%	94.68%
atpg_effectiveness	99.51%	99.49%

#test_patterns	2736
#clock_sequential_patterns	2736
#simulated_patterns	2880
CPU_time (secs)	802.8

Example 2

The following example shows the command flow to identify the redundant faults after simulating external patterns that were created using named capture procedures:

```
set_system_mode analysis
read_patterns my_external_patt
simulate_patterns
identify_redundant_faults
```

Related Topics

[create_patterns](#)

[report_statistics](#)

[simulate_patterns](#)
[set_capture_procedures](#)
[set_fault_type](#)

iForcePort

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iTOPProc that is called during the processing of a procfile. For more information, see [“IJTAG and ATPG in Tessent Shell”](#) in the *Tessent IJTAG User's Manual*.

Allows you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.

Usage

```
iForcePort object_spec {value [ -not_persistent ] | -release } [-after state_transition_list]
```

Description

Allows you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.

An iComparePort or iForcePort command is only allowed in an iTOPProc and not in a normal iPProc as it is not retargetable upward.

It is best to keep the iPProcs that only interact with the ijtag ports separated from the ones using the iForcePort command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iForcePort command is applied on the next [iApply](#) command or the next [iRunLoop](#) command. You can use the -after switch to delay the application of the value to the port and have it appear only after a specific state transition.

An iForcePort command without the -not_persistent switch remains active until one of the following events happen:

- The iForcePort is overwritten with another iForcePort command or — in case of bidirectional ports — with another iComparePort command.
- The iForcePort is explicitly terminated by means of the -release switch.
- The pattern set is closed and the next pattern set is opened without the -no_initial_ireset switch. (Note: this only applies to the context “patterns -ijtag”).
- If an iForcePort command is part of an iTOPProc which is called from within an iMerge block, then the lifetime of the iForcePort command is limited to the retargeting of that iTOPProc.

This is done for the following reason: If there are conflicting iForcePort specifications in concurrent iCalls, the associated iTOPProcs must be retargeted in series. If the

iForcePort specifications survived the end of the iTopProc, the active iForcePort after the iMerge –end would depend on the (undefined) order of the retargeting of the concurrent iCalls. This non-deterministic behavior must be avoided.

You can use this command in an iCall used as part of test_setup for ATPG, however, it makes the iTopProc un-retargetable upward. You should avoid doing so until you reach the chip level.

Arguments

- *object_spec*

A collection containing the top level input or bidirectional port/ports. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to force. Use “read_verilog -interface_only” or “read_vhdl -interface_only” to read in the port list prior to issuing the set_current_design command as shown here:

```
set_context patterns -ijtag -no_rtl  
read_verilog top.v -interface_only  
read_icl top.icl  
set_current_design top
```

If the port is associated to a pulse waveform in the timeplate, you can only force the port to the off value.

Asynchronous clocks cannot be forced to any value.

ICL ports which are part of a ScanInterface cannot be forced to any value.

ICL ports which are involved in the scan path configuration of active parts of the ICL network cannot be forced to any value.

If there is a CT0, CT1 or CTZ constraint on the port, then the iForcePort command can only be used in combination with the matching value.

- *value*

A string representing the value to force as ‘0’, ‘1’, ‘X’ or ’Z’. You can use the 0b prefix to specify a multi-bit binary number or the 0x prefix to specify a hexadecimal number. Just like with the iWrite command, the X value has special meaning. If you use the iForcePort command on a bused port and any of the bits is specified with an X, it means to leave that bit unchanged and preserve its previous value as if the bit was not even part of the iForcePort command as shown here:

```
iForcePort GPIO[3:0] 0b0xx1
```

is equivalent to

```
iForcePort GPIO[3] 0b0  
iForcePort GPIO[0] 0b1
```

- *-not_persistent*

An optional switch that specifies that the forced value can be overwritten by the IJTAG solver if needed during later retargeting operations. This option has no effect when the

iForcePort command is used on a port that is not described in ICL as the solver never interacts with those ports.

- **-release**

An optional switch that specifies that the specified bits are now no longer persistent and the solver is allowed to change their values if needed during future retargeting operations. Using “iForcePort P1 -release” is equivalent to issuing “iForcePort P1 <current_value> -not_persistent” but it does not require you to know the current value.

- **-after state_transition_list**

An optional switch-value pair used to specify a Tcl list of state transitions after which the value is applied to the port. The list of available state transitions are all the ones supported by the IEEE1149.1 state machine, and state names are the same names used in SVF.RESET, IDLE, DRSELECT, DRCAPTURE, DRSHIFT, DREXIT1, DRPAUSE, DREXIT2, DRUPDATE, IRSELECT, IRCAPTURE, IRS SHIFT, IREXIT1, IRPAUSE, IREXIT2, IRUPDATE

Examples

Example 1

The following example forces 2 bits of a bussed port:

```
iForcePort GPIO[3:0] 0b1XX0
iApply
```

It is equivalent to specifying

```
iForcePort GPIO[3] 0b1
iForcePort GPIO[0] 0b0
iApply
```

Example 2

The following example applies the value 1 to GPIO[3] when the state machine goes from Shift-DR to Exit1-DR and the value 0 to GPIO[0] when the state machine goes from Exit1-DR to Update-DR.

```
iForcePort GPIO[3] 0b1 -after DRSHIFT:DREXIT1
iForcePort GPIO[0] 0b0 -after DREXIT1:DRUPDATE
iApply
```

Related Topics

[add_clocks](#)

[iApply](#)

[iComparePort](#)

[iProc](#)

[iProcsForModule](#)

[open_pattern_set](#)

iMerge

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tesson Shell](#)” in the *Tesson IJTAG User's Manual*.

Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.

Usage

iMerge {-begin[-error_on_conflict] | -end}

Description

Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.

iCall is the only PDL command which is allowed in between iMerge -begin and iMerge -end.

In the iProcs called by the iCalls, Tesson IJTAG identifies any resource conflicts, such as two PDL commands writing to the same register, and all conflicts arising from reservations done by means of the command iTake. All conflicting commands are then processed serially. For the remaining iCall commands, Tesson IJTAG tries to find an optimal parallel solution.

Arguments

- **-begin**

A required switch that marks the beginning of the set of iCall commands to be processed in parallel.

- **-error_on_conflict**

An optional switch that specifies to issue an error when there is a conflict rather than process the conflicting events serially. In the case of nested iMerge blocks, you only need to specify this switch on the outermost iMerge specification; the specification applies to all iMerge processes that are called from within the block.

Refer to “[iMerge Conflict Reporting](#)” in the *Tesson IJTAG User's Manual* for more information.

- **-end**

A required switch that marks the end of the set of iCall commands to be processed in parallel.

Return Values

None

Examples

```
iMerge -begin -error_on_conflict
iCall proc1
iCall proc2
iCall proc3
...
iMerge -end
```

Related Topics

[iCall](#)

[iTake](#)

[iPrefix](#)

import_clocks

Contexts: patterns -scan, patterns -scan_retargeting

Mode: setup

Imports functional clocks defined during the pre-DFT design rule checking.

Usage

```
import_clocks [-core core_name] [-verbose]
```

Description

This command imports functional clocks defined during the pre-DFT design rule checking—see “[Pre-DFT Clock Rules \(DFT_C Rules\)](#)” on page 2747.

This command is called in setup mode before you perform design rule checks. Use this command to import functional clocks if you are doing capture with these clocks, and that you already defined during pre-DFT design rule checking.

Arguments

- **-core *core_name***

An optional switch that specify the core name from which functional clocks should be imported. If the -core switch is not specified, the tool attempts to match a core description against the module name of the specified module(s) or instance(s). If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool’s default module matching options which are consistent with what synthesis tools typically use. For information on the default module matching options and how they can be changed, see the description for the “[set_module_matching_options -suffix_pattern_list](#)” option.

If the tool is unable to match a specified module or instance, it issues an error with a reminder that the user can explicitly specify which core description to use with the -core switch.

- **-verbose**

An optional switch that outputs a note to stdout for each clock added by the command.

import_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

Usage

```
import_dfm -rdB rdb_file -layer layer_name -type {cell | open | bridge | cell_open |
    cell_bridge | cell_ton | cell_toff} [-sample n] [-replace]
```

Description

Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

The import_dfm command verifies that the DFM rule violations in the VDB match the layout geometries in the LDB. It then populates the VDB with the DFM rule violations.

Note

 DFM shapes that fail verification are still imported into the VDB and used during annotation.

You must have previously opened the LDB and VDB with the open_layout command. Otherwise, Tessian Diagnosis issues an error message.

For more information, see “[Diagnosis for Design for Manufacturability Analysis](#).”

Arguments

- **-rdB rdb_file**

A required switch and string pair that specifies the Calibre-compatible DFM RDB file.

- **-layer layer_name**

A required switch and string pair that specifies the layer at which the polygons in the RDB file should be compared to diagnosis suspect bounding boxes. The layer name must match one of the layers that appear in the diagnosis report when layout-aware diagnosis and/or cell-aware diagnosis is enabled.

For cell suspects, the layer name is always CELL. For open and bridge suspects, the layer names come from the LEF/DEF files. For cell_open, cell_bridge, cell_ton, and cell_toff suspects, the layer names come from the Tessian CellModelGen UDFM file.

Note

 You can override the default layer names used in the UDFM file. For more information, see the *Tessian CellModelGen Tool Reference*.

-
- **-type {cell | open | bridge | cell_open | cell_bridge | cell_ton | cell_toff}**

A required switch and literal that specifies the type of DFM violation that you want to report, where cell_open, cell_bridge, cell_ton, and cell_toff only apply to cell-aware defects for cell-aware diagnosis. The cell_bridge violation supports inter-layer bridge defects.

- **-sample *n***

An optional switch and string pair that specifies a percentage value from 0 to 100. This value specifies the number of shapes that will be verified against the LDB. Setting this value to 0 prohibits verification.

This switch has no effect on diagnosis. Use it to improve verification runtimes with the caveat that lowering the value decreases the verification accuracy.

- **-replace**

An optional switch that specifies to remove from the VDB the DFM rule with the same rule name, and layer, and to replace it with the new RDB information.

Examples

The following example opens an LDB, creates the directory into which Tesson Diagnosis will write the VDB, and then imports the specified RDB files into the VDB created in the design.ldb directory. For OPEN violations, it verifies 10% of the rule violations in the VDB during verification. In addition, it imports inter-layer bridge violations for layers M0-M1.

```
open_layout design.ldb
import_dfm -fdb id_100.rdb -layer route_2 -type OPEN -sample 10 -replace
import_dfm -fdb inter_layer_bridges_M0_M1.rdb -layer M0-M1 -type CELL_BRIDGE -replace
```

Related Topics

[annotate_diagnosis](#)
[delete_dfm](#)
[report_dfm_rules](#)

import_patterns_from_svf

Context: patterns -ijtag

Mode: analysis

Allows you to integrate the actions described in the SVF file into the currently open pattern set.

Usage

```
import_patterns_from_svf file_name [-scan_interface scan_interface]  
[-maintain_model_state on | off]
```

Description

Allows you to integrate the actions described in the SVF file into the currently open pattern set.

The import_patterns_from_svf command can only be used if a pattern set is currently open. The appropriate actions from the SVF file are appended to the existing retargeted actions. The internal state of the ICL network is updated accordingly. All comments in the SVF file are converted into iNote statements unless the comment starts with !!, in which case they are ignored.

The import_patterns_from_svf command can be used either if there is a complete ICL description without any blackboxes, or if the top module is a blackbox.

import_patterns_from_svf cannot be used if one of the instances, which are accessed by the SVF patterns, is a blackbox.

An ICL blackbox module is a module which contains only ports, scan interfaces and organizational elements like parameters, attributes, enums and aliases, but none of the following:

- DataRegister or ScanRegister
- DataMux, ClockMux or ScanMux
- LogicSignal
- OneHotDataGroup or OneHotScanGroup
- Instance
- AccessLink
- The Source property of an output port
- The Enable property of an output port

If the imported SVF file contains SIR or SDR commands, and you create retargeted PDL with the "write_patterns -pdl" command, then the output PDL will contain illegal references to dummy register names (tesson_blackbox_data_register, tesson_blackbox_instruction_register)

if the top module is a blackbox. These register names can also be found in the results of the get_pattern_set_data command.

The import_patterns_from_svf command cannot be used in an iMerge block or in an iProc. At the time when this command is used, there must not be any queued iWrite, iRead, iForcePort or iComparePort commands.

The import_patterns_from_svf command has the following limitations:

- The SVF import must not leave the TAP controller in RESET state.
- ENDDR IRPAUSE and ENDDR RESET are not supported. The SVF command ENDDR can be used with the arguments DRPAUSE and IDLE only.
- ENDIR DRPAUSE and ENDIR RESET are not supported. The SVF command ENDIR can be used with the arguments IRPAUSE and IDLE only.
- The SVF command RUNTEST can only be used in its simplest form:
RUNTEST *run_count* [TCK|SCK].
The *run_state*, *min_time*, *max_time* and *end_state* arguments are not supported.
- The commands RUNTEST, SDR and SIR cannot be used if the TAP controller is in RESET state.
- The STATE command does not accept several arguments, that is, it is not possible to specify the TAP state transitions the TAP controller should choose to get to the specified end state. Only the end state itself can be specified.
- TRST Z is not supported. TRST only accepts ON, OFF and ABSENT.
- Native SVF does not have a possibility to express the measurement of high impedance at an output port. Therefore write_patterns –svf creates a dedicated annotation for that purpose: “!TESSENT_PRAGMA expect_z”. This annotation is not understood by import_patterns_from_svf, it is simply ignored.

Arguments

- *file_name*
A required string that specifies the name of a SVF file to be read. The file can be gzip-compressed or uncompressed
- *-scan_interface scan_interface*
An optional switch and string pair that specifies the name of the TAP scan interface in the ICL top module. If this switch is omitted, there must be a unique TAP scan interface in the ICL top module. A scan interface is identified as a TAP scan interface by the existence of a TMS port.

Note

 A pattern set can have active scan interfaces and inactive scan interfaces. If there are several TAP scan interfaces, but only one of them is active, then the `-scan_interface` switch can be omitted.

- `-maintain_model_state on | off`

An optional switch and literal pair that specifies how the SVF reader operates. When set to off, the reader operates in “blackbox mode”, in that it will not apply the SDR (Scan Data Register) and SIR (Scan Instruction Register) commands to the internal state of the IJTAG network, nor will it check if the internal state of the IJTAG network matches the length of the provided SDR/SIR data. The default value is on.

Examples

The following example converts an SVF file to a PDL file:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl data/icl/*.icl
SETUP> set_current_design chip
SETUP> add_clocks ClkA -pulse_always -period 10ns
SETUP> set_system_mode analysis
ANALYSIS> open_pattern_set test1 -tester_period 100ns
ANALYSIS> import_patterns_from_svf jtagtest1.svf
ANALYSIS> close_pattern_set
ANALYSIS> write_patterns jtagtest1.pdl -pattern_sets test1 -pdl
ANALYSIS> exit -f
```

Related Topics

[iComparePort](#)
[iForcePort](#)
[iMerge](#)
[iProc](#)
[iRead](#)
[iWrite](#)

import_scan_mode

Context: dft -edt, dft –edt –logic_bist, dft –test_points, patterns -scan

Mode: setup

Imports configuration settings from tcd_scan.

Usage

```
import_scan_mode [-core core_name] [scan_mode_name] [-no_reset_design]  
[-fast_capture_mode {on | off}] [-verbose] [-silent]
```

Description

Imports the following configuration settings from tcd_scan:

- Adds scan chains used in the scan mode.
- Adds EDT instruments used in the scan mode.
- Adds OCC instruments used in the scan mode.
- Configures scan clocks used in the scan mode.
- Configures DFT signals used in the scan mode.
- Adds scan clock pulses to shift procedure.
- Adds scan enable signal forces to shift procedure.

Note

 To override default parameter values of imported scan instruments, use the [set_core_instance_parameters](#) command.

By default, the tool resets the design when you issue the import_scan_mode command unless you specify the -no_reset_design switch. In addition, the tool uses an auto generated [test procedure file](#).

Arguments

- **-core *core_name***

An optional switch that specifies the core name from which the scan mode should be imported. If the -core switch is not specified, the tool attempts to match a core description against the module name of the specified module(s) or instance(s). If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool's default module matching options which are consistent with what synthesis tools typically use. For information on the default module matching options and how they can be changed, see the description for the “[set_module_matching_options -suffix_pattern_list](#)” option.

If the tool is unable to match a specified module or instance, it issues an error with a reminder that the user can explicitly specify which core description to use with the **-core** switch.

- **scan_mode_name**
An optional string that specifies the scan modes to import. This switch is only required when there are multiple scan modes. If only one scan mode is specified in the TSDB, it is imported by default.
- **-no_reset_design**
An optional switch that specifies that the design should not be reset. By default, the design is reset when the **import_scan_mode** command is issued.
- **-fast_capture_mode on | off**
An optional switch that specifies using fast capture mode. The default is off.
- **-verbose**
An optional switch that displays additional information about the configuration settings being imported.
- **-silent**
An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example overrides an EDT instrument parameter on all EDT instruments imported with the **import_scan_mode** command:

```
import_scan_mode edt_mode
set_core_instance_parameters -instrument_type edt \
    -parameter_values {edt_low_power_shift_en on}
```

Example 2

This example does not reset the design and imports the design in the only mode that the TSDB has:

```
import_scan_mode -no_reset_design
```

For designs with multiple scan modes, you need to specify the *scan_mode_name* to import.

Example 3

This example imports the design in *scan_mode_name* “*edt_mode*”. The tool uses an auto generated test procedure file and resets the design. The **-verbose** switch shows additional configuration settings being imported.

```
import_scan_mode edt_mode -verbose
```

Example 4

This example shows how the `import_scan_mode` command uses the `int_mode` as the `scan_mode_name`. Additionally, the example demonstrates how to use a specific test procedure file with the `set_procfile_name` command, which you specify after issuing the `import_scan_mode` command:

```
import_scan_mode int_mode -verbose  
set_procfile_name my_custom.testproc
```

Related Topics

[report_scan_modes](#)
[set_core_instance_parameters](#)

index_collection

Context: unspecified, all contexts

Mode: all modes

Extracts a single object at a specified index in a collection and creates a new collection containing only that object.

Usage

`index_collection collection index`

Description

Extracts a single object at a specified index in a collection and creates a new collection containing only that object.

The base collection remains unchanged.

Arguments

- ***collection***

A required string that specifies the collection to be searched.

- ***index***

A required integer that specifies the index into the collection. Allowed values are from 0 to `sizeof_collection - 1`.

Examples

The following example shows the first and last object in a collection being extracted.

```
set c1 [get_instances {u1 u2 u3}]
{u1 u2 u3}

get_name_list [index_collection $c1 0]
u1

get_name_list [index_collection $c1 [expr [sizeof_collection $c1] -1 ]]
u3
```

Related Topics

[add_to_collection](#)

[append_to_collection](#)

[compare_collections](#)

[copy_collection](#)

[filter_collection](#)

[foreach_in_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

iNote

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Inserts a note in the opened pattern set.

Usage

iNote *string*

Description

Inserts a note in the opened pattern set.

The note is attached to the time corresponding to where the iNote command is executed. When the pattern set is written to a pattern file using the [write_patterns](#) command, the note will appear as an annotation in the pattern file at the appropriate location. For simulation test benches, the note is displayed when the cycle containing the annotation is simulated.

Using verilog_insert pragma Within an iNote String

The following pragma can be used as the start of the iNote string argument to insert native Verilog code inside the Verilog test bench. The Verilog code specified in the pragma is executed at the time when the iNote annotation would normally be inserted. See “[Example 2](#).”

The pragma uses the following format:

iNote “tessentPragma verilog_insert *native verilog statement suitable within an initial block*”

The native Verilog statement following the pragma must be code that is allowed to go inside an initial block. This code must be terminated with a semicolon or the end keyword following the normal Verilog grammar rules. “[Example 2](#)” shows the insertion of a while loop and the triggering of an event.

Arguments

- *string*

The comment string or note that will be added to the pattern set at the proper time. This must be a single normal Tcl string, Multiple words must be enclosed in quotes or braces.

Return Values

None

Examples

Example 1

In the following example, the first iNote command is attached to the beginning of the pattern. The iNote string contains a variable that will be substituted before being sent to the iNote command. The second iNote command is attached to the time corresponding to the time reached to execute the previous iApply actions. The last iNote appears after the loop.

```
iNote "Setting up controller instance $inst"
iWrite mode blue; iApply
iNote "Running controller"
iWrite enable On; iApply
iRunLoop 30 -tck
iNote "Checking results"
iRead results pass; iApply
```

Example 2

The following example makes use of the special insert_verilog tessent_pragma (shown in green in the example code block) to perform two tasks. As previously described, the text that follows the “tessent_pragma insert_verilog” prefix in the iNote string is native Verilog code that is suitable for being inserted inside an initial Verilog block. The example uses one special iNote to insert the following line at a specific time in the test bench:

```
while (dut.cm pll.lock !== 1'b1) #100;
```

This code while construct makes the test bench wait for the lock output pin on the PLL instance to become 1 before continuing with the rest of the operation. As described in the comments in the dofile example, this is used to mimic the effect of a STIL MatchLoop that would be used on the tester.

The second special iNote in the example is to trigger a Verilog event in the test bench:

```
-> check_clock${::clock_id_cnt}_period;
```

The event that is triggered by this command lets the Verilog code that checks the period of the clock to do its jobs. See the “always @ (check_clock`(id)_period)” statement in the check_clock_string defined in the dofile. This string is processed by the “string map” Tcl command to replace the ‘(id), ‘(path) and ‘(period) values before being sent to a file. That file is later included into the test bench using the [SIM_MODULE_INCLUDE](#) parameter in the [write_design](#) command. Notice that the [SIM_INSTANCE_NAME](#) parameter is also used in the write_design command so that the instance name of the chip within the test bench is constant and can be used in the Verilog code inserted into the test bench. The “dut” string in the instance path shown above is the instance name supplied using the [SIM_INSTANCE_NAME](#) parameter.

```
set_context patterns -ijtag
read_design my_chip -view interface
set_current_design my_chip

add_clocks clk -period 9.5ns

set_system_mode analysis

# This section contains a Verilog code block that will be processed by the
# Tcl "string map" command to replace '(id)', '(path)' and '(period)' by
# specific values. See string map usage in the check_clock iProc below.

set check_clock_string {
    real clock^(id)_edge1_time, clock^(id)_edge2_time,
        measured_clock^(id)_period;
    event check_clock^(id)_period;

    always @(check_clock^(id)_period) begin
        # Measure the realtime for the next two clock edges
        @ (posedge dut.^{path}) clock^(id)_edge1_time = $realtime;
        @ (posedge dut.^{path}) clock^(id)_edge2_time = $realtime;
        measured_clock^(id)_period = clock^(id)_edge2_time -
            clock^(id)_edge1_time;
        # Increment the compare event copunter
        tb._compare_count = tb._compare_count + 1;
        $timeformat(-9, 3, " ns", 10);
        if (1.01*measured_clock^(id)_period < ^{period} || \
            0.99*measured_clock^(id)_period > ^{period}) begin
            # Increment the miscompare counter when the period is not within
            # 1% of the expected period.
            tb._compare_fail_count = tb._compare_fail_count+1;
            $display(
                " Clock '{path}' has wrong period: expected %0t, actual %0t",
                ^{period}, measured_clock^(id)_period
            );
        end else begin
            $display(" Clock '{path}' has expected period of %0t",
                measured_clock^(id)_period
            );
        end
    end
}
# This iProc is used to reset the PLL
iProcsForModule pll
iProc reset {} {
    iWrite rst 0b1
    iApply
    iWrite rst 0b0
    iApply
}
```

```

# This iProc is used to check that the PLL is locked.
iProc check_lock {} {
    iRead lock 0b1
    iApply
    iClock ref
    iClock vco
    set icl_scope [get_icl_scope -icall]
    # Use the "hierarchical_design_instance" attribute to map the ICL pin
    # into the verilog design. Use to_verilog_path helper proc to convert
    # slashes into dots.
    set verilog_scope [::mentor::to_verilog_path \
        [lindex [get_attribute_value_list $icl_scope
            -name hierarchical_design_instance] 0]]
    # Use get_clock_option to get expected period of PLL input and output
    # pins from the ICL description and the add_clocks period specified as
    # top of the dofile
    set ref_period \
        [expr {round([get_iclock_option ${icl_scope}.ref -period
            -in ns]*1000)/1000.0}]
    set vco_period \
        [expr {round([get_iclock_option ${icl_scope}.vco -period
            -in ns]*1000)/1000.0}]
    # Use the echo command to take result of "string map" and save it to a
    # file so that it can be inserted into the test bench. See
    # "write_patterns -parameter_list" option below where SIM_MODULE_INCLUDE
    # is used. The content of the ::check_clock_string is define above.
    echo [string map [list `$(id) $::clock_id_cnt \
        `(path) ${verilog_scope}.ref \
        `(period) $ref_period \
        $::check_clock_string] > check_clock_set1.v
    # Initialize ::clock_id_cnt if it does not already exists
    if {! [info exists ::clock_id_cnt]} {set ::clock_id_cnt 0}
    # Use iNote pragma to trigger the check of the clock period
    iNote "tesson_pragma verilog_insert \
        -> check_clock${::clock_id_cnt}_period;"
    incr ::clock_id_cnt
    # Repeat same method for the vco output
    echo [string map [list `$(id) $::clock_id_cnt \
        `(path) ${verilog_scope}.vco \
        `(period) $vco_period \
        $::check_clock_string] >> check_clock_set1.v
    iNote "Measuring clock period of ${verilog_scope}.vco"
    iNote "tesson_pragma verilog_insert \
        -> check_clock${::clock_id_cnt}_period;"
    incr ::clock_id_cnt
}
# Create two pattern sets. In simulation they are written out together but
# in STIL they will written out separately so that P2 which checks the
# lock #of the PLL can be used inside a STIL MatchLoop.
open_pattern_set p1
    iCall cm_pll.reset
    # Use iNote pragma to insert while loop in verilog test bench to
    # imitiate the behavior of the MatchLoop. It checks the lock pin every
    # 100ns until it becomes 1.
    iNote "tesson_pragma verilog_insert \
        while (dut.cm pll.lock !== 1'b1) #100;"
    iCall cm_pll.check_lock
    iRunLoop -time 100ns

```

```
close_pattern_set -network_end_state initial

# Uses SIM_MODULE_INCLUDE parameter to have the file created above
# included into the test bench.
# Note that is it no longer needed to specify the SIM_INSTANCE_NAME
# parameter because the macro SIM_INSTANCE_NAME is added in the
# testbench and is used in the iNote above to get the DUT instance
# name of the chip: `SIM_INSTANCE_NAME.
write_patterns pll_setup.v -repl \
    -verilog \
    -pattern_sets {p1 p2} \
    -param_list {SIM_MODULE_INCLUDE check_clock_set1.v}
```

insert_test_logic

Context: dft -scan, dft -test_points

Mode: analysis

Prerequisites: To use the *-Verilog* switch, you must have defined a buffer model with the [add_cell_models](#) command or with a [cell_type](#) attribute.

Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.

Usage

Context: dft -test_points

```
insert_test_logic [-write_in_tsdb { on | off | auto }]
    [-design_identifier design_identifier]
    [-write_insertion_script script_file_name]
    [-replace]
    [-insertion [dc | rc]]
```

Context: dft -scan

```
insert_test_logic [-write_in_tsdb { on | off | auto }]
    [-design_identifier design_identifier] [-replace]
```

Description

Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.

The tool inserts test structures into a design to increase the fault coverage (testability) of the design. Specifically, the tool attempts to increase a design's fault coverage with the `insert_test_logic` command by doing the following:

- Identifies sequential cells and replaces them with corresponding scan cells, and stitches them together into a scan chain. (Referred to as scan cell replacement and stitching.)
- Supports the adding of both system-defined and user-defined test points.
- Supports the automatic adding of test logic.

For more information on scan replacement, test points and test logic, refer to [Test Point Analysis and Insertion](#) in the *Tessent Scan and ATPG User's Manual*. For information on handling pre-inserted scan cells, refer to “[How to Specify Existing Scan Cells](#)” in the *Tessent Scan and ATPG User's Manual*.

You can also use the `insert_test_logic` command with the `-write_insertion_script` switch to generate insertion scripts for test points for Design Compiler (DC) or RTL Compiler (RC). For more information, see “[How to Insert Test Points and Perform Scan Stitching Using Third-Party Tools](#)” in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-write_in_tsdb on| off | auto**

An optional switch and literal pair that controls whether the extracted ICL is automatically written out in the specified TSDB output directory.

The auto value means “on” when the directory returned by [get_tsdb_output_directory](#) already exists, otherwise auto means “off.”

When set to “on,” the DFT inserted design will be written out into the following directory:

```
tsdb_output_directory/dft_inserted_designs/  
module_name_design_id.dft_inserted_design
```

The *design_id* value is the one returned by [get_context -design_id](#). See the [dft_inserted_designs](#) section of the [Tessent Shell Data Base \(TSDB\)](#) chapter for more information. If you have an elaborated ICL view in memory, it will be exported to the new DFT inserted design directory with an updated tessent_design_id attribute value matching the current design id. The concatenated PDL file will also be copied from its original source.

- **-design_identifier *design_identifier***

An optional switch and value pair that sets the *design_identifier*.

Note



This switch will be ignored if you have previously set the *design_identifier* using the “[set_context -design_identifier *string*](#)” command.

- **-write_insertion_script *script_file_name***

An optional switch that enables the script generation feature to generate the insertion script for inserting test points for Design Compiler (DC) or RTL Compiler (RC).

- **-replace**

An optional switch that specifies to replace the output file generated by the insertion script, if the file already exists.

- **-insertion dc | rc**

An optional switch that specifies the format of the target insertion tool language of the insertion script. The default target for the insertion script is Design Compiler (DC).

Related Topics

[add_scan_instances](#)

[add_tied_signals](#)

[analyze_wrapper_cells](#)

[report_scan_chains](#)

[set_test_logic](#)

[set_scan_signals](#)

set_test_point_insertion_options

intercept_connection

Context: dft

Mode: insertion

A command that uses the `get_dft_cell` command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.

Usage

```
intercept_connection node [-cell_function_name function_name]
[-technology dft_cell_selection_name]
[-input2 input2_source] [-select select_source]
[-leaf_instance_prefix prefix_name]
[-only_when_has_functional_source]
```

Description

A command that uses the `get_dft_cell` command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.

This command performs interception inside both unqualified and not unqualified modules.

Arguments

- **node**

A required value that is either a port, pin, or net name in the design or a collection of one port, pin, or net object. Based on the type of object and its direction attribute, in the case of a pin or a port, the source or destination connection of the object is intercepted.

[Table 4-13](#) shows the interception that is performed based on the node type. Note that an error is generated if the node is an inout pin or port, or if the node is a net corresponding to an inout pin or port.

Table 4-13. Interception based on node type

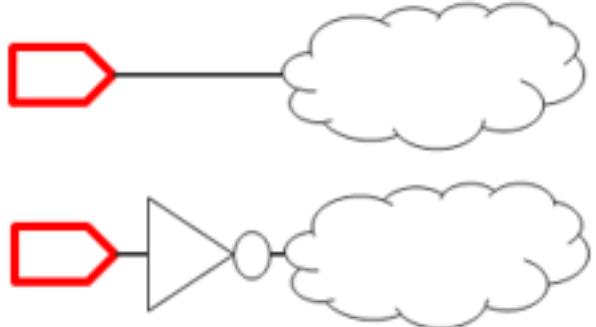
Node Type	Interception Done	Illustration
Input port	The node is a driver. The intercepting cell ends up between the input port and its original fanout.	

Table 4-13. Interception based on node type (cont.)

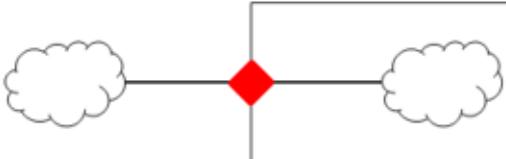
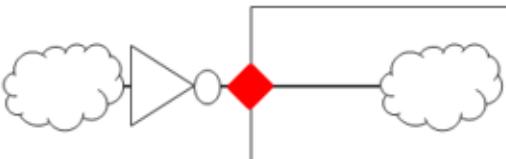
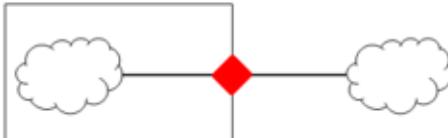
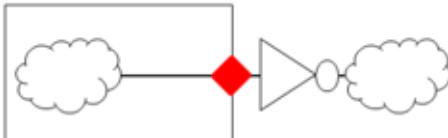
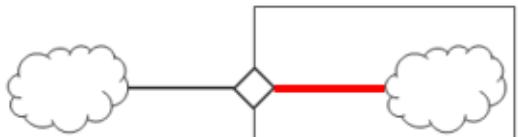
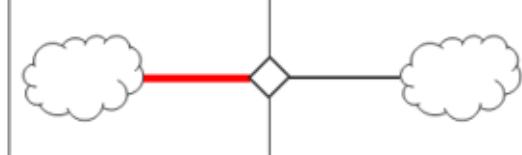
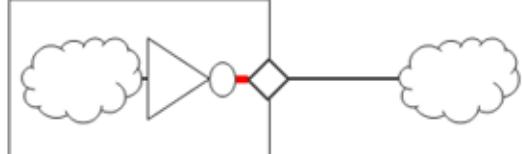
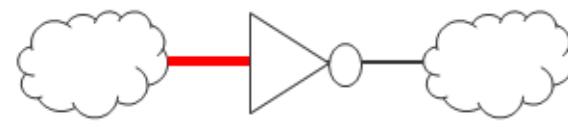
Node Type	Interception Done	Illustration
Output port	The node is a sink. The intercepting cell ends up between the original fanin and the output port	Before and after interception:  
Input pin	The node is a sink. The intercepting cell ends up between the original fanin and the input pin	Before and after interception:  
Output pin	The node is a driver. The intercepting cell ends up between the output pin and its original fanout.	Before and after interception:  
Net corresponding to an input pin or an input port	The node is a driver. The intercepting cell ends up between the input pin or port and its original fanout.	Before and after interception:  

Table 4-13. Interception based on node type (cont.)

Node Type	Interception Done	Illustration
Net corresponding to an output pin or an output port	The node is a sink. The intercepting cell ends up between the original fanin and the output pin or port.	Before and after interception:  
Net not corresponding to a pin or a port	The node is a driver. The intercepting cell ends up between the driver of the net and its original fanout. Only single driver nets can be intercepted.	Before and after interception:  

- cell_function_namefunction_name**

A required switch and string pair that specifies the function of the cell to be used for the interception. [Table 4-14](#) shows the allowed cell function names and how the cell function affects the need to specify the -input2 and -select options.

Table 4-14. Allowed Cell Function Names

Function Name	Results
and	The interception is performed with the 2 input AND cell returned by the “get_dft_cell and” command. The -input2 option is required to specify the source of the second input. In the <i>Tessent Cell Library Manual</i> , see AND Cells for more details about the “and” cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option.
buffer	The interception is performed with the BUFFER cell returned by the “get_dft_cell buffer” command. In the <i>Tessent Cell Library Manual</i> , see Buffer and Input Pad for more details about the “buffer” cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option.

Table 4-14. Allowed Cell Function Names (cont.)

Function Name	Results
inverter	The interception is performed with the INVERTER cell returned by the “get_dft_cell inverter” command. In the <i>Tessent Cell Library Manual</i> , see Inverter for more details about the “inverter” cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option.
mux	The interception is performed with the MUX cell returned by the “get_dft_cell mux” command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the <i>Tessent Cell Library Manual</i> , see MUX Cell for more details about the “mux” cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
or	The interception is performed with the 2 input OR cell returned by the “get_dft_cell or” command. The -input2 option is required to specify the source of the second input. In the <i>Tessent Cell Library Manual</i> , see OR Gate for more details about the “or” cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option.
clock_and	The interception is performed with the 2 input clock AND cell returned by the “get_dft_cell clock_and” command. The -input2 option is required to specify the source of the second input. In the <i>Tessent Cell Library Manual</i> , see AND Cells for more details about the “clock_and” cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function <code>clock_in</code> , it is selected for the interception.
clock_buffer	The interception is performed with the clock BUFFER cell returned by the “get_dft_cell clock_buffer” command. In the <i>Tessent Cell Library Manual</i> , see Buffer and Input Pad for more details about the “clock_buffer” cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.
clock_inverter	The interception is performed with the clock INVERTER cell returned by the “get_dft_cell clock_inverter” command. In the <i>Tessent Cell Library Manual</i> , see Inverter for more details about the “clock_inverter” cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.

Table 4-14. Allowed Cell Function Names (cont.)

Function Name	Results
clock_mux	The interception is performed with the MUX cell returned by the “get_dft_cell clock_mux” command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the <i>Tessent Cell Library Manual</i> , see MUX Cell for more details about the “clock_mux” cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
clock_or	The interception is performed with the 2 input clock OR cell returned by the “get_dft_cell clock_or” command. The -input2 option is required to specify the source of the second input. In the <i>Tessent Cell Library Manual</i> , see OR Gate for more details about the “clock_or” cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function clock_in, it is selected for the interception.

• **-technology *dft_cell_selection_name***
An optional switch and value pair that specifies from which dft_cell_selection list the cell is to be used. You only need to specify this option if your library has more than one dft_cell_selection wrapper. If you have none, one is learned for the simple cell types, but you will not have access to the clock cells unless you have set their cell_type attribute in the library. See “[Model-Level Attribute Descriptions](#)” in the *Tessent Cell Library Manual* for information on how to specify the cell_type attribute.

• **-input2*input2_source***
An optional switch that specifies a node to connect to the second input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins at some later time. [Table 4-14](#) on page 1091 describes which values for -cell_function_name makes the specification of this option mandatory.

• **-select *select_source***
An optional switch and value pair that specifies a node to connect to the select input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins at some later time. [Table 4-14](#) describes which values for -cell_function_name make the specification of the -select option mandatory.

• **-leaf_instance_prefix *prefix_name***
This optional switch and value pair specifies the prefix to used when instantiating the cell. When this is not specified, the default is “insertion_”. The leaf instance name is constructed by concatenating the prefix_name with the cell_function_name. When a conflict is detected during instantiation, the tool automatically adds a “_<int>” suffix to uniquify the leaf name.

You can use a specific prefix if you want to easily find the inserted cells in a synthesis or layout script by using a command such as “`get_cells prefix_name -hier`”.

- `-only_when_has_functional_source`

This optional switch specifies to skip the instantiation of the cell and interception of the node when the attribute `has_functional_source` is 0 for the node. Instead, the node is disconnected and driven by the specified `-input2` value. You cannot specify this option when `-cell_function_name` refers to a single input cell as there is no `-input2` value to connect to in that case.

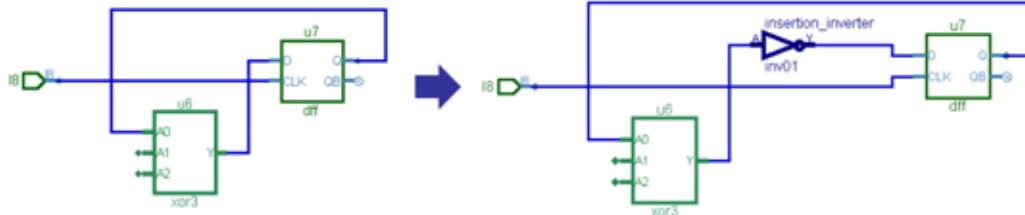
Examples

Example 1

The following example inserts a simple inverter gate by intercepting the existing connection between the u7/D pin and the u6/Y pin, shown on the left side of [Figure 4-6](#). After the interception, the inverter A pin is connected to the u6/Y pin and the inverter Y pin is connected to the u7/D pin as shown on the right side of [Figure 4-6](#).

```
INSERTION> intercept_connection u7/D -cell_function_name inverter  
{insertion_inverter}
```

Figure 4-6. Inverter Interception



Example 2

The following example inserts a multiplexer and specifies to connect the second multiplexer input pin to 1, and to connect the multiplexer select pin to `scan_en`.

```
INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 \  
-select scan_en  
{u4/insertion_mux_1}
```

This example does the same thing as first example above but specifies the `-only_when_has_functional_source` option. Because this option is specified, the tool connects the pin u1/u3/D directly to a constant 1 if its `has_functional_source` attribute evaluates to 0. In this case, the command returns null as no intercepting cell was instantiated.

```
INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 \  
-select scan_en -only_when_has_functional_source  
{}
```

Example 3

The following example script shows how to cascade three gates on a set of nodes. The example illustrates how the command is capable of handling non-unique designs. The script will work whether module “rep3” is unqualified or not. Notice how the example inserts the first gate level on all nets, and then adds two more levels in a row on each net. This illustrates that it does not matter if you insert all gate levels on each net or if you add one level at a time. In both cases, the result is identical.

Original design:

```
module ex (ina, outa, c1, c2, c3);
    input c1, c2, c3;
    input [3:0] ina;
    output [3:0] outa;
    rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]));
    rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]));
    rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]));
    rep1 rep1 (.ina(ina[3]), .outa(outa[3]));
endmodule
module rep3 (ina, outa);
    input ina;
    output outa;
    assign outa = ina;
endmodule
module rep1 (ina, outa);
    input ina;
    output outa;
    assign outa = ina;
endmodule
```

The dofile:

```
set nets [get_nets -of_pins [get_pins -direction input]]
foreach_in_collection net $nets {
    intercept_connection $net -cell or -input2 c1}
foreach_in_collection net $nets {
    foreach var {2 3} {
        intercept_connection $net -cell [lindex {and or} [expr $var-2]] \
            -input2 c$var -leaf_inst_prefix myprefix_
    }
}
```

The modified design:

```
module ex (ina, outa, c1, c2, c3);
    input c1, c2, c3;
    input [3:0] ina;
    output [3:0] outa;
    rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]), .c1(c1), .c2(c2), .c3(c3));
    rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]), .c1(c1), .c2(c2), .c3(c3));
    rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]), .c1(c1), .c2(c2), .c3(c3));
    rep1 rep1 (.ina(ina[3]), .outa(outa[3]), .c1(c1), .c2(c2), .c3(c3));
endmodule
module rep3 (ina, outa, c1, c2, c3);
    input ina;
    output outa;
    input c1, c2, c3;
    wire c1, c2, c3;
    wire ina_ts1, ina_ts2;
    OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));
    AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));
    OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule
module rep1 (ina, outa, c1, c2, c3);
    input ina;
    output outa;
    input c1, c2, c3;
    wire c1, c2, c3;
    wire ina_ts1, ina_ts2;
    OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));
    AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));
    OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule
```

Related Topics

[get_dft_cell](#)
[move_connections](#)

iOverrideScanInterface

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Imposes user-specified behavior on the operation of a ScanInterface.

Usage

```
iOverrideScanInterface scan_interface_list [-capture {on | off}] [-update {on | off}]  
[-broadcast {on | off}]
```

Description

Imposes user-specified behavior on the operation of a ScanInterface.

The iOverrideScanInterface command imposes user-specified behavior on the following three aspects of the operation of a ScanInterface:

- Whether or not it allows the capture of new data into the scan registers connected to that interface.
- Whether or not it allows the update of new data shifted into the scan registers connected to that interface.
- Whether or not it participates in scan broadcasting.

Arguments

- *scan_interface_list*

A required list of ICL ScanInterfaces to which the following settings will apply. Only ordinary ScanInterfaces are allowed, no TAP ScanInterfaces, because the TAP state machine does not allow omitting the Capture-DR/Capture-IR cycle or the Update-DR/Update-IR cycle.

The ScanInterfaces must be interfaces of the current design. ScanInterfaces of internal instruments are not supported yet.

- -capture {on | off}

An optional switch and literal pair that specifies whether or not the affected scan registers will capture data from their parallel inputs, that is, from their specified CaptureSource. If this switch is set to off, the retargeting software ensures that the affected registers cannot be used to observe iRead targets on their parallel inputs. Either a different possibility to observe the target can be found or the retargeting software issues an error message.

The “affected registers” are those registers which are on the path between the ScanInPort of a specified client ScanInterface and the corresponding ScanOutPort of that ScanInterface.

At startup time, the capture functionality is switched on for all scan interfaces. This default setting is reapplied by the iReset command and when a pattern set is opened without the -no_initial_ireset switch.

- **-update {on | off}**

An optional switch and literal pair that specifies whether or not the affected scan registers pass on their serially loaded data to their parallel outputs. If this switch is set to off, the retargeting software ensures that the affected registers cannot be used to justify a certain value at their parallel outputs. Either a different possibility to justify the value can be found, or the retargeting engine issues an error message.

At startup time, the update functionality is switched on for all scan interfaces. This default setting is reapplied by the iReset command and when a pattern set is opened without the -no_initial_ireset switch.

- **-broadcast {on | off}**

This optional switch is currently not supported and does not have any effect.

Return Values

None.

Examples

The following example disables capture on scan interface client1.

iOverrideScanInterface client1 –capture off

The following example disables update on two scan interfaces:

iOverrideScanInterface c2 c3 –update off

iPDLLevel

Context: all contexts

Mode: all modes

Identifies the level and version of PDL commands.

Usage

iPDLLevel *level* [-version *version*]

Description

Identifies the level and version of PDL commands.

The iPDLLevel command is mandated by the IEEE 1687-2014 standard to be the first command in the PDL file, and it specifies that all the subsequent commands are PDL commands of the specified level and version.

Note

 Tessent Shell does not verify the compliance of subsequent commands with the specified PDL level. Tessent Shell accepts Level-0 PDL as well as Level-1 PDL, independent of the specification within the iPDLLevel command.

Arguments

- *level*
A required value that specifies the PDL level. The allowed values are 0 and 1.
- -version *version*
An optional switch and string pair that specifies the PDL version. Currently, the only allowed value is STD_1687_2014.

Return Values

None.

iPrefix

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Sets the iPrefix path that is used to compute the effective_icl_instance_path for the [iCall](#) command and other PDL commands.

Usage

iPrefix [icl_instance_path]

Description

Sets the [iPrefix path](#) that is used to compute the effective_icl_instance_path for the [iCall](#) command and other PDL commands.

The effective_icl_instance_path is described in detail in the [iCall](#) command description.

This command is not affected by the previous iPrefix command. If the icl_instance_path argument is omitted, the [iPrefix path](#) value is set to empty. If the path is specified and the iPrefix command is *inside* an iProc, it must be a valid instance path relative to the current [iProc instance path](#). If the path is specified and the iPrefix command is *not* inside an iProc, it must be a valid instance path relative to the current design.

Arguments

- *icl_instance_path*

An optional string that defines the new value for [iPrefix path](#) used to compute the effective_icl_instance_path for the [iCall](#) command and other PDL commands such as the [iRead](#), [iWrite](#), and [iClock](#) commands.

Return Values

None

Examples

In the following example, the [iPrefix path](#) value is first set to block1_i1.inst1_i1. It is then set to block1_i2 with the second invocation of the iPrefix command. Notice how the second iPrefix command completely replaces the value of the iPrefix path and is not influenced by the previous value of iPrefix path. Only the [iCall](#), [iClock](#), [iRead](#), [iWrite](#), [iRunLoop](#), and [iClockOverride](#) commands are affected by the iPrefix command.

iPrefix block1_i1.inst1_i1
iWrite A 0b0
iPrefix block1_i2
iWrite inst1_i1.A 0b0

Related Topics

[iCall](#)
[iClock](#)
[iClockOverride](#)
[iRead](#)
[iWrite](#)

iProc

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The [iProcsForModule](#) has been specified.

Specifies a PDL procedure that can run later when referenced by the iCall command.

Usage

iProc ***name variables body***

Description

Specifies a PDL procedure that can run later when referenced by the [iCall](#) command.

Analogous to the Tcl command “proc”, the specification consists of the procedure name, the procedure variables, and the procedure body.

The specified procedure is registered against a specific ICL module that must have been previously loaded by [read_icl](#) and specified by [iProcsForModule](#). Since the iProc is registered against a certain module, several iProcs with the same name may exist, each associated to a different ICL module. If an iProc is specified with a name that already exists for the current [iProcsForModule](#), the new specification replaces the existing one. If an ICL module is deleted, the related iProcs are deleted too.

When the procedure is called using the iCall command, the variables in the argument list are substituted with the values specified in the iCall command. The commands inside the iProc body are executed in the scope of the instance against which the iProc is called.

Arguments

- ***name***

Name of the iProc which must be a simple scalar identifier. A simple scalar identifier starts with a letter and contains only letters, decimal digits, and underscores.

- ***variables***

A Tcl list of variable name or variable-value pairs enclosed in curly braces. The list of variables can contain either simple identifiers or identifier-value pairs, where the value specifies the default value of the variable. All simple identifiers must be in the list before the identifier-value pairs.

- ***body***

Commands to be executed when the procedure is called by means of the iCall command.

Return Values

None

Examples

The following example shows an iProc called run_testa that is registered against the myInstrument1 ICL module. The iProc has two arguments: cnt and mode. The mode argument is optional because it has a specified default value of “red”. The body of the iProc contains standard Tcl syntax and PDL commands. The cnt argument is specified before the mode argument because it has no specified default value.

```
iProcsForModule myInstrument1

iProc run_testa {cnt {mode red}} {
    set cycles(red) 50
    set cycles(blue) 90
    set cycles(green) 70
    iRead go Fail
    iRead done No
    iApply
    iWrite mode $mode
    if {$mode == "red"} {
        iWrite p1 0b0
        iWrite p1(2) 0b1
    }
    iApply
    iWrite enable Yes
    iApply
    iRunLoop $cycles($mode) -tck
    iRead done Yes
    iRead go Pass
    iRead count $cnt
    iApply
}
```

Related Topics

[delete_iprocs](#)
[get_iproc_argument_list](#)
[get_iproc_list](#)
[iCall](#)
[iProcsForModule](#)
[iTopProc](#)

iProcsForModule

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The corresponding ICL module already loaded.

Specifies the ICL module for which subsequent iProc commands refer to.

Usage

iProcsForModule *icl_module_name* [-iProcNameSpace *name_space*]

Description

Specifies the ICL module for which subsequent iProc commands refer to.

An iProc is a PDL procedure written for a specific ICL module.

All references to ICL objects within the iProc refer to the instance path relative to the module it is written for. Once you have issued the iProcsForModule command, all iProcs defined afterward are registered against the specified ICL module. You cannot define any iProc until the iProcsForModule is set.

The iProcsForModule is specified once for each ICL module that you have iProcs to register against it. The iProcsForModule can only be specified for ICL modules that already have been loaded using the [read_icl](#) command.

To be sure that you load iProcs against the correct module, you can store all iProcs associated to a given ICL module inside a single file and add the proper iProcsForModule at the top of the file. With this organization, you can load the iProcs for any module as soon as the specified ICL module has been read in using the [read_icl](#) command.

Arguments

- *icl_module_name*

Name of a valid ICL module which has already been loaded into memory using the [read_icl](#) command.

- -iProcNameSpace *name_space*

An optional switch and string that creates a sub name space within each name space created from the ICL module name. When *name_space* is not specified, it defaults to the value set by the [iUseProcNameSpace](#) command.

Return Values

None

Examples

In the following example, the file myCore.icl is read in; the file contains the entire ICL description of myCore. A child module of myCore is myInstrument1. The file *myInstrument1.iprocs* contains an iProcsForModule command that specifies that myProc defined afterward is meant to be registered against this module.

Sourcing the file *myInstrument1.iprocs* loads the iProcs defined in it. One of the instances of myInstrument1 within myCore is u2.u3. The iCall u2.u3.myProc command executes the iProc called myProc defined for module myInstrument1 against the u2.u3 ICL instance of that module. The solver will retarget the PDL found within that iProc to the boundary of myCore.

```
read_icl myCore.icl
source myInstrument1.iprocs
set_current_design myCore
set_system_mode analysis
open_pattern_set P1
iCall u2.u3.myProc
close_pattern_set
```

myInstrument1.iprocs file contains

```
iProcsForModule myInstrument1
iProc myProc {} {
    iRead u1.R 0b0
    iApply
}
```

Related Topics

[delete_iprocs](#)
[get_iproc_argument_list](#)
[get_iproc_list](#)
[iCall](#)
[iProc](#)

iPulseClock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

In the “patterns –ijtag” context, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iTopProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Allows you to start or restart a synchronous clock that previously has been forced to its off-state by means of the [iForcePort](#) command.

Usage

`iPulseClock object_spec`

Description

Inside an IJTAG pattern set, iPulseClock can be used to start or restart a synchronous clock that previously has been forced to its off-state by means of the [iForcePort](#) command.

An iPulseClock command is only allowed in an iTopProc and not in a normal iProc as it is not retargetable upward. It is best to keep the iProcs that only interact with the ICL ports separated from the iTopProcs using the iPulseClock command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iPulseClock command is applied on the next [iApply](#) command or the next [iRunLoop](#) command.

Arguments

- *object_spec*

A collection containing the top level clock/clocks that is/are going to be restarted. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to pulse. Furthermore, the port must have been specified as a clock port by means of the [add_clocks](#) command. Use “`read_verilog -interface_only`” or “`read_vhdl -interface_only`” to read in the port list prior to issuing the `set_current_design` command as shown here:

```
set_context patterns -ijtag -no_rtl
read_verilog top.v -interface_only
read_icl top.icl
set_current_design top
add_clocks 0 clkRZ
add_clocks 1 clkRO
```

Asynchronous clocks and internal clocks cannot be used as an argument of the iPulseClock command. Constrained ports or ports which have not been subject to an [add_clocks](#) command cannot be used, either.

Examples

The following example forces two clocks to their off-states, waits for 1000 TCK cycles and writes a value to an ICL port. Then it starts the clocks pulsing, waits for 2000 TCK cycles and writes another value to the ICL port.

```
set_context patterns -ijtag
read_icl top.icl
read_verilog top.v
set_current_design top
add_clocks 0 clk1
add_clocks 1 clk2
set_system_mode analysis

open_pattern_set patset1 -tck_ratio 4
  iForcePort {clk1 clk2} 0b01
  iRunLoop 1000
  iWrite din 0b1
  iApply
  iPulseClock {clk1 clk2}
  iRunLoop 2000
  iWrite din 0b0
  iApply
close_pattern_set
```

Related Topics

[add_clocks](#)
[iApply](#)
[iForcePort](#)
[iRunLoop](#)
[iTopProc](#)
[open_pattern_set](#)

iRead

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Adds a read operation to the command queue that will be solved by the next iApply command.

Usage

iRead [*icl_instance_path.*]*readable_icl_object* [*value*]

Description

Adds a read operation to the command queue that will be solved by the next iApply command.

You can specify an optional expected value to which the read value will be compared to when observed. The [iApply](#) command description describes how the iRead commands are interpreted by the iApply command.

Arguments

- [*icl_instance_path.*]*readable_icl_object*

Specifies the ICL objects that are to be read. You can specify to add read operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed *effective_icl_instance_path*. The *effective_icl_instance_path* is described in detail in the [iCall](#) command description.

- *value*

An optional constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the *readable_icl_object* in the ICL description.

Return Values

None

Examples

The following example requests the *readable_icl_objects* done and go to be read and compared against the value represented by the enumeration symbol called Yes and Pass defined in ICL. The ICL object *count* is scheduled to be read but not compared to any expected value. The three requested read operations are solved when the [iApply](#) command executes.

iRead done Yes
iRead go Pass
iRead count
iRead status 0b1
iApply

Related Topics

[iApply](#)

[iCall](#)

[iPrefix](#)

[iWrite](#)

iRelease

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tesson Shell](#)” in the *Tesson IJTAG User's Manual*.

Releases a resource previously taken by means of iTake.

Usage

`iRelease [icl_resource]`

Description

iRelease allows the release of the ownership of a resource that has been reserved with iTake.

After iRelease of the resource, the resource may be freely altered upon merging with other iProcs, and the retargeting software can also make use of the resource again, for example, to reconfigure the scan network or to justify observation paths.

Arguments

- *icl_resource*

An optional value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* will be released.

If you issue the command without this argument, the effect is the same as if an empty string was passed. In this case, the effective instance is the one that is determined by the ICL instance path of the currently processed iProc and by the iPrefix command. All registers and ports of this instance are released.

Return Values

None.

Examples

iRelease RegA
iRelease

Related Topics

[iCall](#)

[iMerge](#)

[iTakes](#)

iReset

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.

Usage

iReset [-sync]

Description

Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.

The exact sequence performed depends on the presence of specific ports in the current design. Each ResetPort is pulsed high and each TRSTPort is pulsed low. If there are TMSPorts without a corresponding TRSTPort, TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low.

After an iReset command, the entire ICL is in its reset state and the TAP finite state machines (when present) are in RunTestIdle state.

You should only use the iReset command once at the beginning of a pattern set. It is illegal to include an iReset inside an instrument PDL as its effect is global and destroys any ongoing action from all instruments. The iReset command is only allowed after [open_pattern_set](#), in an iProc associated with the current design, or in test_setup/test_end procedures (the latter only allowed in contexts other than "patterns -ijtag"). Placing it inside an instrument-level iProc prevents this iProc from being iCalled from a higher level module.

If there is no explicit iReset at the beginning of an open pattern set, an iReset is implied unless the -no_initial_ireset option is used with the [open_pattern_set](#) command. Refer to the [open_pattern_set](#) command description to understand when and how the -no_initial_ireset option should be used.

Arguments

- -sync

An optional switch that forces a synchronous reset even if there are TRSTPorts in the current design. TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low. If there is no TMS port, the -sync switch is ignored.

Return Values

None

Examples

The following examples set the ICL network into the reset state:

```
iReset  
iReset -sync
```

iRunLoop

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Creates a vector loop of a given duration.

Usage

```
iRunLoop {count [-tck | -sck [icl_instance_path.]icl_clock_port]} | {-time tvalue}
```

Description

Creates a vector loop of a given duration.

In the generated pattern, the loop is converted into a loop of tester periods.

All clocks defined with the [add_clocks](#) commands toggle at their specified period in a free running fashion during the entire pattern. When using the -tck option, the iRunLoop count is expressed in terms of TCK periods and TCK is toggling during the loop. When the -sck [icl_instance_path.]icl_clock_port option is used, TCK is not toggling.

The specified [icl_instance_path.]icl_clock_port must have been specified in a prior [iClock](#) command such that its controlling source was verified and its effective cumulative **multiplication factor computed**. The size of the loop in the pattern depends on the computed effective multiplication factor as well as the ratio between the clock source period and the tester period. [Table 4-15](#) outlines the resulting loop size in the generated pattern based on the various options.

Table 4-15. RunLoop Scaling Algorithm

Condition	Results
open_pattern_set -tck_ratio = 1 iRunLoop X -tck	TCK is defined with a pulsing waveform with the falling edge preceding the rising edge. The loop count is X and is over a single vector where TCK is set to pulse
open_pattern_set -tck_ratio = 2 iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over 2 vectors where TCK is low for the first vector and high for the second one

Table 4-15. RunLoop Scaling Algorithm (cont.)

Condition	Results
open_pattern_set -tck_ratio = 4, 8, 16,... iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over <tck_ratio> vectors where TCK is low for the first and last quarter of vectors and high for the middle ones.
iRunLoop X -sck <clk> where <clk> has a source that was defined as an asynchronous clock using the “add_clocks -period” command	TCK is inactive during the loop. The loop count is Y and is over a single vector. $Y = X / \text{<effective multiplication factor>} * (\text{<async clock period>} / \text{<tester period>})$
iRunLoop X -sck <clk> where <clk> has a source which was defined as a synchronous clock and the waveform defined for this clock has 2 edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. $Y = X / \text{<effective multiplication factor>}$
iRunLoop X -sck <clk> where <clk> has a source that was defined as a synchronous clock and the waveform defined for this clock has 2*n edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. $Y = X / \text{<effective multiplication factor>} * 1/n$

Arguments

- **count**
Required integer that specifies the number of clock cycles to be seen on TCK or on [icl_instance_path.]icl_clock_port. It is not required when the -time switch is used.
- **-tck**
Optional switch that specifies that count refers to the test clock TCK.
- **-sck [icl_instance_path.]icl_clock_port**
Optional switch and value pair that specifies for which clock pin the count is for. The value [icl_instance_path.]icl_clock_port specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the [iCall](#) command description. An [iClock](#) command on the [icl_instance_path.]icl_clock_port must have been issued prior to issuing the iRunLoop command.
- **-time tvalue**
Optional switch and value pair that specifies a time delay *tvalue* instead of a cycle count. When TCK is OFF, iRunLoop -time is converted to a single vector loop, with a loop count

of *tvalue*/*tester_period* rounded up to the next integer. If TCK is asynchronous free-running, TCK is ON and the loop count must be a multiple of the TCK ratio.

Return Values

None

Examples

Example 1

The following example specifies a run loop of 10000 tck period. The tck clock is toggling during the loop when the iRunLoop command is using the -tck option.

iRunLoop 10000 -tck

Example 2

The following example specifies a run loop of a 1000 system clock cycles on the clk pin of the u1 instance. The iClock command checks that the clock port is controlled by a valid source and computes the cumulative multiplication factor between the primary input and the clock port. The run loop is automatically adjusted such that the u1.clk port receives a minimum of 1000 cycles at the u1 instance. The TCK port may or may not be toggling.

iClock u1.clk

iRunLoop 1000 -sck u1.clk

Example 3

The following examples specify run loops that must last for at least the specified time.

iRunLoop -time 2000ns

iRunLoop -time 4.56E+2ns

iRunLoop -time 5.67E-1us

Related Topics

[add_clocks](#)

[delete_clocks](#)

[iClock](#)

[iClockOverride](#)

[report_icl_modules](#)

[report_clocks](#)

[ALL_MAX_LOOP](#)

is_collection

Context: unspecified, all contexts

Mode: all modes

Indicates whether string is a collection handle.

Usage

```
is_collection string [-with_object obj_type] [-single_type_objects]
```

Description

Indicates whether string is a collection handle.

This command returns true if the provided string is a collection handle. You use this command inside a Tcl proc to check if the passed in argument is a collection of objects versus names of objects. See the example below for an example usage.

You can also use this command to check if the collection contains only one type of objects or objects of a specific type.

Arguments

- ***string***

A required string that specifies the collection whose existence is to be verified. This command returns true if the collection exists.

- **-with_object *obj_type***

An optional switch and string that specifies whether the collection specified by *string* contains at least one object of the type specified by *obj_type*. This command returns true if at least one object of the type specified by *obj_type* exists.

- **-single_type_objects**

An optional switch that specifies whether the collection specified by *string* contains objects of all the same type. This command return true only if all objects are of the same type.

Examples

The following example uses the `is_collection` command to verify that the `net` argument is already a collection of nets and, if not, the `get_nets` command is used to create the collection of nets assuming `<net>` is instead a Tcl list of net names. If the passed-in argument is a collection, it is checked to verify that it only contain net objects. With this code, the proc `myProc` can be invoked with either a Tcl list of names of nets, or a collection of nets and the proc automatically adapts.

```
proc myProc {net} {
    if {[is_collection $net]} {
        if {![[is_collection $net -with_object net -single_type_objects]]} {
            return -code error "<net> contains other object types than net"
        }
    } else {
        set net [get_nets $net]
    }
    #$net is definitely a collection of nets
    ...
}
```

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[compare_collections](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

iTakē

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Takes ownership of ICL resources to prevent the retargeting software from altering their states during the processing of subsequent iApply commands or during the processing of concurrent iProcs in iMerge parallelization.

Usage

`iTake [icl_resource] [-persistent]`

Description

An iTakē command followed by an iWrite command to the associated taken resource permanently prevents Tessent IJTAG from altering the value during the processing of subsequent iApply commands. This is particularly useful if the affected resources are also involved in the configuration of the scan paths or in the justification of observation paths, in which case the retargeting software usually changes the states in its sole discretion. A state on a port or the content of a register that is taken by an iTakē command becomes unchangeable by the retargeting software after the iProc that executes an iWrite to that port or register. Prior to being taken and written, the retargeting software is free to change the value (for example, to reconfigure the scan network or to justify observation paths). Once written by an iWrite command in the PDL, the constraint can be lifted only by an iRelease command for that resource. The resources remain taken until the end of the iProc which contains the iTakē command, if they are not explicitly released by means of iRelease before.

In the context of an iCall command that is to be merged with other iCall commands by means of iMerge, an iTakē command applies the same restrictions to the resources as in the single PDL command stream, but in addition, it reserves the resource for exclusive use by the iProc containing the iTakē. Once an iProc has taken a resource, values assigned to the resource with iWrite are not allowed to be altered by concurrent iCalls until the resource is explicitly released with iRelease. Clock frequencies on ports which are taken as a resource cannot be changed by concurrent iCalls, either.

Arguments

- *icl_resource*

An optional value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* will be taken.

If you issue the command without this argument, the effect is the same as if an empty string was passed. In this case, the effective instance is the one that is determined by the ICL

instance path of the currently processed iProc and by the iPrefix command. All registers and ports of this instance are taken.

- -persistent

An optional switch that specifies that the iTake is not released at the end of the iCall that invoked it. The iTake will carry forward into the following pattern set if it was created with the “open_pattern_set -no_initial_reset” command.

Use this switch when you want to set a value on an object and prevent the solver from changing that value in the subsequent iCall. For example, if you want to fix an object to a value in an iCall referenced by a [ProcedureStep](#) wrapper, and you want the effect to persist in the following [ProcedureStep](#) and [TestStep](#) sequences, use “iTakes –persistent” in the first iProc otherwise its effect will be cleared once the iProc is done and will not carry over to the next iCalls.

Return Values

None.

Examples

```
iTake RegA  
iTakes
```

Related Topics

[iCall](#)

[iMerge](#)

[iPrefix](#)

iTopProc

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The [iProcsForModule](#) has been specified.

Specifies a PDL procedure that can run later when referenced by the iCall command, when its associated ICL module is the top level module.

Usage

`iTopProc name variables body`

Description

The iTopProc command is identical to the iProc command except that it can only be iCalled against the top level ICL module. An iTopProc is never retargetable upward.

The purpose of an iTopProc is to define an iProc that allows having the iForcePort and iComparePort commands inside it. Those two commands are not retargetable upward, therefore they cannot be used inside an iProc that is meant to be retargetable.

You can only iCall an iTopProc when the associated ICL module is the current ICL top module, that is, when it matches the current design. You cannot iCall an iTopProc from within an iProc. However, you can iCall an iProc or an iTopProc inside an iTopProc.

Arguments

- ***name***

Name of the iProc which must be a simple scalar identifier. A simple scalar identifier starts with a letter and contains only letters, decimal digits, and underscores.

- ***variables***

A Tcl list of variable name or variable-value pairs enclosed in curly braces. The list of variables can contain either simple identifiers or identifier-value pairs, where the value specifies the default value of the variable. All simple identifiers must be in the list before the identifier-value pairs.

- ***body***

Commands to be executed when the procedure is called by means of the iCall command.

Return Values

None

Related Topics

[delete_iprocs](#)

[get_iproc_argument_list](#)

[get_iproc_list](#)

[iCall](#)

[iProc](#)

[iProcsForModule](#)

iUseProcNameSpace

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.

Usage

`iUseProcNameSpace [name_space]`

Description

Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.

The scope of this command is limited to within the iProc in which it appears. Invoking this command a second time within the same iProc changes the name space to the new value of *name_space*.

If iUseProcNameSpace is invoked without the *name_space* argument, the name space is set to that of the root module.

Arguments

- *name_space*

An optional string that specifies the name space.

The name space name must be formed from an initial upper or lower case alphabetic character followed by an optional sequence of upper or lower case alphanumeric or underscore characters ([a-zA-Z][a-zA-Z0-9_]*)�.

Return Values

None

Examples

```
iUseProcNameSpace MyNameSpace //set namespace to MyNameSpace  
iUseProcNameSpace // return to root name space
```

Related Topics

[get_iproc_argument_list](#)

[get_iproc_list](#)

[iCall](#)
[iProc](#)
[iProcsForModule](#)

iWrite

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the [open_pattern_set](#) command.

In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “[IJTAG and ATPG in Tessent Shell](#)” in the *Tessent IJTAG User's Manual*.

Adds a write operation to the command queue that will be solved by the next iApply command.

Usage

iWrite [*icl_instance_path.*]**writable_icl_object** *value*

Description

Adds a write operation to the command queue that will be solved by the next iApply command.

The ICL objects that can be written are DataRegisters, ScanRegisters, DataOutPorts, DataInPorts or an alias of any of these objects. A value to be written must be specified as the second argument. How the iWrite commands are interpreted by the [iApply](#) command is described in the [iApply](#) command description.

Arguments

- [*icl_instance_path.*]**writable_icl_object**

Specifies the ICL objects that are to be written. You can specify to add write operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the [iCall](#) command description.

- ***value***

An optional constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the written objects in the ICL description.

The X value has special meaning. If you use the iWritePort command on a bused port and any of the bit is specified with an X value, it means to leave that bit unchanged and preserve its previous value as if the bit was not even part of the iWrite command as shown here

```
iWrite DI[3:0] 0b0xx1
```

is equivalent to

```
iWrite DI[3] 0b0  
iWrite DI[0] 0b1
```

Return Values

None

Examples

The following example requests the writable_icl_object mode to be written with the value represented by the enumeration symbol goNogo defined in ICL. The two requested write operations are solved when the iApply command is executed.

```
iWrite mode goNogo
iWrite Count 0xa6
iApply
```

Related Topics

[iApply](#)

[iCall](#)

[iPrefix](#)

[iRead](#)

launch_sid_tester

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Directs Tesson Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file.

Usage

`launch_sid_tester -cdp cdp_path [-new]`

Arguments

- **-cdp *cdp_path***

A required switch and string pair that specifies a CDP.

- **-new**

An optional switch that specifies to create a new CDP at the location specified by the -cdp switch. By default, the tool opens an existing CDP.

Description

Directs Tesson Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file. Additionally, this command specifies an existing CDP or initializes a new CDP.

Note

 You can only work with one SID tester at a time. To launch a new SID tester you must first shutdown the current SID tester with the shutdown_sid_tester command.

During the launch, Tesson Shell passes the hardware and pin map configuration data to the SID tester so that the SID tester can perform the proper setup tasks.

You must have loaded the configuration data file with the read_config_data command prior to running this command.

Note

 SVF-based CDPs are configured with asynchronous clocking. The tool does not provide clock waveforms to the DUT, therefore you must source the appropriate clocks yourself.

Examples

The following example initializes a new CDP named “cdp1” and launches the SID tester process.

`launch_sid_tester -cdp ./cdp1 -new`

Refer to “Executing Non-Tessent IJTAG Instruments” in *Tessent SiliconInsight User’s Manual for Tessent Shell* for a sample usage scenario.

Related Topics

[read_config_data](#)

[shutdown_sid_tester](#)

lock_current_registration

Context: unspecified, all contexts

Mode: all modes

Locks all previously registered commands and attributes preventing them from being unregistered and modified.

Usage

lock_current_registration

Description

Locks all previously registered commands and attributes preventing them from being unregistered and modified.

After you verified that your registered commands and/or attributes work correctly, you can make them into a *plugin* that Tesson Shell automatically loads during initialization. Simply place your files in a directory called *plugin_dir/tcl_modules/my_dir* where *plugin_dir* and *my_dir* are arbitrary directory names. Set the TESSENT_PLUGIN_PATH environment variable to point to the *plugin_dir* directory and any files with a *.tcl* extension found inside *plugin_dir/tcl_modules/my_dir* will be evaluated into the Tcl namespace ::*tcl_modules*::*my_dir*. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon.

If you name your *plugin_dir* directory name “tesson_plugin” and place it one level above the bin directory of Tesson Shell, it will be automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the command lock_current_registration is issued so that they cannot be unregistered accidentally.

Arguments

None

Examples

The following example registers a string attribute called myAtt for the object type pin and then issues a lock_current_registration command to make it impossible to unregister the attribute.

```
register_attribute -name myAtt -value_type string -object_type pin  
lock_current_registration
```

Related Topics

[register_tcl_command](#)

[register_attribute](#)

[unregister_attribute](#)

[unregister_tcl_command](#)

macrotest

Context: dft -edt, patterns -scan

Mode: analysis

Generates manufacturing test patterns for embedded logic and memories referred to as macros.

Usage

```
macrotest {-MULtiple_macros macro_filename}  
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]  
[-VERIFY | -NO_VERIFY] [-MASK_nonobservation_sites]  
[-NO_FILL_patterns] [-TE_observation_only | -LE_observation_only]  
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]  
[-FAILure_analysis] [-EXHAUSTIvely_search]  
[-SKIP_PATterns num_to_skip] [-FIRST_OBS_pattern]
```

or

```
macrotest {{ID# | pin.pathname | instance_name} pattern_filename}  
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]  
[-VERIFY | -NO_VERIFY] [-MASK_nonobservation_sites]  
[-NO_FILL_patterns] [-TE_observation_only | -LE_observation_only]  
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]  
[-FAILure_analysis] [-EXHAUSTIvely_search]  
[-SKIP_PATterns num_to_skip] [-FIRST_OBS_pattern]
```

Description

Runs the MacroTest utility to automatically generate manufacturing test patterns for embedded logic and memories referred to as macros. This utility is useful for applying specific patterns that test the insides of a macro, thereby improving overall IC test quality.

Note

 Although a macro is typically embedded logic or a memory, it can also be a disjoint set of internal sites or a single block of hardware represented by an instance in HDL.

MacroTest reads patterns from the text file you provide and converts them into scan-based manufacturing test patterns. These test patterns, when applied to the chip, deliver the underlying patterns to the appropriate embedded location through intervening logic. The following is a brief summary of the capabilities of MacroTest:

- Has no impact on area or performance
- Enables you to test multiple macros in parallel
- Supports user-selected scan observation points
- Allows you to define macro output values that do not require observation

- Supports synchronous memories; for example, supports positive (or negative) edge-triggered memories embedded between positive (or negative) edge-triggered scan chains

As a special form of the run command, the macrotest command uses all existing ATPG and cell constraints and limits.

Note

 To obtain a high degree of precision for resolving any conversion problems, MacroTest simulates one pattern at a time, rather than 32 at a time.

The observability of some patterns may not be verified. Pin(s) and pattern are reported and whether or not observability was verified, and then processing stops. Typically, this is because a scan chain clock and an input on the macro are shared. Also, pulsing the macro pin pulses the scan clock, causing the scan cells to update, making the observability invalid. In such cases, specify a sequential MacroTest pattern such that you specify the macro input values to read, including a pulse of the shared clock, in the first cycle of the pattern. Specify the known output values in the second cycle. MacroTest uses the sequential engine for such patterns, allowing a full cycle for the values to come out of the macro and propagate to the scan cells where they will be captured. Such operation of a shared clock should let you verify the macro's observability.

If the macro being tested is a RAM, the MacroTest patterns might perform several writes followed by several reads. In this situation, each write and read becomes a separate scan pattern with its own scan load. If loading the scan chain causes spurious writes into the RAM, Tesson FastScan destroys the data, making the reads access invalid data rather than the data that was written; therefore, the macro should be able to hold state while operating the scan chain to allow a sequence of inputs to be converted to a valid sequence of scan patterns (with a scan load per scan pattern).

For the rare case where macro outputs have paths to both leading edge and trailing edge capturing scan cells, the macrotest command includes switches that let you specify either trailing edge or leading edge observation. When you prepare the MacroTest patterns file, you can specify when to capture or observe so that the pattern matches the selection. For more information, see the `-Te_observation_only` and `-Le_observation_only` switches.

For more information about using the macrotest command, see “[MacroTest Overview](#)” in the Tesson Scan and ATPG User’s Manual.

Note

 When testing multiple macros, each macro opens a pattern file. When the number of open files reaches your operating system-specific limit, no more files can be opened. This may force MacroTest to fail. To avoid this problem, you may need to issue the appropriate version of `limit`, `unlimit`, or `ulimit` Linux shell commands before running MacroTest to increase the maximum number of files that can be open at one time. Check your Linux documentation for the appropriate command.

Arguments

- **-MULTiple_macros *macro_filename***

A switch and string pair that test multiple macros in parallel using the macrotest commands in the *macro_filename* file. The file must contain only macrotest commands, one for each of the macros to test in parallel. Mentor Graphics recommends that you first test each macro individually in a separate run. Individual runs allow analysis if success is not possible for any single macro. When you test each macro successfully, add its corresponding command to *macro_filename* and remove it from the dofile (or command line).

Note

 If you successfully tested a macro individually, you can probably test it successfully in parallel. However, Mentor Graphics cannot guarantee this due to the enormous variety of possible embeddings.

If you use the -Multiple_macros switch, you must include in the file all macros you want to be simultaneously tested. You cannot use this switch and also specify a pattern_filename or individual macro (by its ID#, pin.pathname or instance_name) on the same command line. You can test one subset of all macros using one -Multiple_macros run, with one macro_filename containing those macros and their pattern file specifications, and then test another subset by using another -Multiple_macros run. If you use other arguments with the -Multiple_macros switch, then macrotest uses those arguments as the default arguments for all macros in the file. For an example that uses this switch, see the Examples section at the end of this command description.

- ***ID#***
An integer that specifies the gate identification number of the object to use in the macrotest run. The value of the *ID#* argument is the unique identification number that is automatically assigned to a gate within the design during the model flattening process.
- ***pin.pathname***
A string that specifies the name of the input or output pin of a Tesson Cell library model to use in the macrotest run.
- ***instance_name***
A string that specifies the name of the instance to use in the macrotest run.
- ***pattern_filename***
A string that specifies the name of the file containing the set of patterns to be applied to the macro.
- **-L_H**
An optional switch that represents {L,H} as {LO,HI} output values in the macro patterns file. An error is issued if a 0 or 1 output value is specified in the macro patterns file. This is the default.

- **-NO_L_H**

An optional switch that uses {0,1} to specify {LO,HI} output values in the patterns file, rather than the default {L,H}. If the default {L,H} is used, checking is done to ensure that the pin direction matches (output pin for L,H; input pin for 0,1). If the option -NO_L_h is issued, no such checking is possible.

- **-VARY_observe_by_pattern**

An optional switch that, if issued, causes MacroTest to assume that a macro may need to vary its observation path or change where it is observed at after the first successful observation path is discovered by MacroTest on the first pattern with known outputs. The default is to assume the same path can be reused without calling ATPG to verify that on each pattern.

- **-VERBose**

An optional switch that causes all informative messages to be issued. Use this option when testing any macro for the first time, so that all information about the progress and possible issues are conveyed. This is the default.

- **-NO_VERBose**

An optional switch that turns off the default verbose output. When using this option, important warning and informative messages may be left out.

- **-VERIfy**

An optional switch that causes one extra simulation per pattern to verify each macro's controllability (the macro input values) and observability (the values captured into a scan cell or observed at a chip pin). The observability check verifies that complementing all macro outputs causes each scan cell or chip pin where observation is occurring to change its value if a defect caused the observed macro output to be faulty. This is the default, and you should never turn off this check to solve problems of failure to verify a pattern, which will halt the macrotest run. Instead, you must debug the issue causing the failure. Failure to verify control values typically causes simulation or tester mismatches. Failure to observe typically means that defects in the macro cannot be detected on a tester, which defeats the entire purpose of creating macrotest patterns.

- **-NO_VERIfy**

An optional switch that refrains from performing an extra simulation per pattern to verify that the macro input values were delivered by the pattern, and that changing the macro outputs changes the observation sites. This switch should never be used rather than investigating the cause of verification failure. Failure to control or observe the macro will either make the pattern useless, or will predict the wrong value and halt the tester.

- **-MASK_nonobservation_sites**

An optional switch that places an X in all scan cells except the ones used for observation on a pattern-by-pattern basis.

You need to specify both the -NO_FILL_patterns and the -MASK_nonobservation_sites switches to force only the scan cells used by macrotest to observe macro outputs to have

known values for the scan unload. If you specify the `-NO_FILL_patterns` switch without also specifying the `-MASK_nonobservation_sites` switch, all values that propagate to scan cells are captured, regardless of whether they are from the macro outputs or from the known PIs and scan cells that fanout to both the macro inputs and other scan cells; that is, scan cells other than those observing macro outputs may have known scan-out values.

- `-NO_FILL_patterns`

An optional switch that causes the unspecified bits in scan chains to remain at a value X, so that only the values needed to test the macro appear as known values. Using this option, you can see which scan chain and primary input values are needed to test the macro by using the [write_patterns](#) command. You can use the `-MASK_nonobservation_sites` option to convert captured values that are not needed but known to X in the scan-out.

- `-TE_observation_only`

An optional switch that observes at only one of the trailing edge (TE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- `-LE_observation_only`

An optional switch that observes at only one of the leading edge (LE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- `-NO_Report_observation_candidates`

An optional switch that refrains from reporting the reachable observation sites. In some cases, the number of possible scan cells that an output can go to is large and the report long, so the default is not to report the candidates.

- `-Report_observation_candidates`

An optional switch that reports the reachable observation sites, and whether those sites are already eliminated as observation candidates due to inability to capture. For example, a cell constraint to observe X prevents capture. If you use this switch, a list of possible observation sites (candidates) per macro output pin is displayed at the beginning of the macrotest run. The `-Report_observation_candidates` switch is sometimes useful for discovering that different macros in a multiple macro run have the same observation site possibilities. It also helps explain why it may be impossible to observe both macros at the same time.

The `observe_at` syntax in the macrotest patterns file allows you to specify where to observe one or more of the outputs of a macro for all patterns. The `-Report_observation_candidates` switch is useful for selecting the name(s) to use as observation site(s).

Note

 To quickly test many macros at once, correct `observe_at` guidance can reduce the runtime. Incorrect guidance leads to failure to succeed where success might have been possible. Sometimes, guidance is necessary to obtain macrotest patterns in a reasonable runtime.

- **-FAILure_analysis**

An optional switch that specifies the tool to determine why ATPG cannot succeed. Because this failure analysis can consume significant processing time, the default is to avoid failure analysis for rapid debug iteration. You should use this switch only as a last resort to determine why the embedding prevents justification of the macrotest pattern that causes ATPG to fail.

- **[-EXHAUSTIvely_search]**

An optional switch that performs an exhaustive search. For some situations, an exhaustive search is needed; however, MacroTest can run for cpu weeks when hundreds of macros are tested together in one run. This switch only solves problems where one macro interacts with another macro; this switch is not used for problems where a single macro conflicts with its own needed boundary values and/or netlist requirements surrounding that macro. By default, this switch is omitted due to the infrequency of such debilitating macro interactions and the typical runtime cost to find them if they occur.

- **-SKIP_PATterns *num_to_skip***

An optional switch and integer pair that skips the first *num_to_skip* patterns. This option is used for analyzing problems after pattern creation when a MacroTest run fails. For example, if the third pattern fails to be converted (the transcript shows two patterns converted and simulated and then reports a failure), specify 2 for *num_to_skip*, so that MacroTest skips the first two patterns and starts on the third pattern. This enables you to quickly iterate on the problematic pattern. The **-SKIP_PATterns *num_to_skip*** option is not used for production patterns.

- **-FIRST_OBS_pattern**

An optional switch used only for analysis when MacroTest fails. This option skips all of the control-only patterns and processes the first pattern with known (observable) macro outputs as if it were the first pattern in order to speed iterations and get to the problematic patterns sooner.

Examples

The following example tests all of the macros in parallel (those with a macrotest command in the “macrofile” file). The example changes the default, whereby output 0 values in MacroTest patterns are indicated by L, and output 1 values are indicated by H. Instead, 0 and 1 are used respectively. Note that macrotest commands inside the macrofile can override the **-NO_L_H** switch below by selecting an alternative option. For example, one or more of the commands could use **-L_H** if those macros have files that use L and H to indicate 0 and 1 output values.

```
macrotest -multiple_macros macrofile -NO_L_H
```

For more examples, refer to “[MacroTest Overview](#)” in the *Tessent Scan and ATPG User’s Manual*.

Related Topics

[add_faults](#)

[set_decision_order](#)
[set_contention_check](#)
[set_gate_report](#)
[set_macrotest_options](#)

mark_display_instances

Context: all contexts

Mode: setup, analysis

Changes the color of the specified instances in the schematic windows.

Usage

```
mark_display_instances {gate_id# | instance_name | pin.pathname}... | -All | -Selected}  
[-marking_index {1 | 2 | 3 | 4 | 5}] [-Display {FLAT_schematic | HIEarchical_schematic}]
```

Description

Changes the color of the specified instances in the schematic windows.

You can specify up to five different colors to use in marking. The color difference can help you to quickly locate and identify particular instances in a complex schematic view. The color difference remains until you use the [unmark_display_instances](#) command to return the gate to its unmarked color.

Tip

 If a gate is selected, the color applied by the mark_display_instances command is not visible until the gate is unselected.

You can change the colors used for marking by choosing **Edit > Preferences** in the DFTVisualizer and clicking the **Colors** tab. For more information, refer to “[Setting DFTVisualizer Preferences](#)” in the *Tessent Shell User’s Manual*.

When you mark an object(s) by entering the mark_display_instances command into the command line (as opposed to using menus or Ctrl-M), the color assigned to Color Index 1 in the DFTVisualizer Preferences dialog box is used to mark the object(s). By default, Green is assigned to Color Index 1. For more information, refer to “[Colors Preferences Dialog Box](#)” in the *Tessent Shell User’s Manual*.

Arguments

- **gate_id#**

A repeatable integer that specifies the gate identification number of an instance to mark. The value of the *gate_id#* argument is the unique identification number assigned to every gate in the design during the model flattening process.

- **instance_name**

A repeatable string that specifies the name of an instance to mark.

- **pin.pathname**

A repeatable string that specifies the name of a pin to mark.

- **-All**
A switch that specifies to mark all the gates displayed.
- **-Selected**
A switch that specifies to mark all currently selected gates.
- **-marking_index {1| 2 | 3 | 4 |5}**
An optional switch and literal pair that specifies the color to be used to mark objects in DFTVisualizer windows. By default, index 1 is used to mark objects. You can specify the color associated with each index from the Colors tab of the DFTVisualizer Preferences dialog box when Marked is selected. For more information on marking, see “[Customizing Marking Colors in the Schematic Windows](#)” and “[Marking and Unmarking Objects in the Windows](#)”.
- **-Display {FLAt_schematic | HIErarchical_schematic}**
A switch and literal pair that specifies the window for which to apply the `mark_display_instances` command. The following literals can be used with the -Display switch:
 - FLAt_schematic—Applies the settings to the Flat Schematic window. This is the default behavior if the -Display switch is not utilized.
 - HIErarchical_schematic—Applies the settings to the Hierarchical Schematic window.

Examples

The following example marks two instances using the color set as Color Index 3, specifying the instance name of one and the gate identification number (85) of the other:

```
mark_display_instances /i$142/q 85 -marking_index 3
```

Related Topics

[select_display_instances](#)
[unmark_display_instances](#)

merge_cdp

Context: patterns -silicon_insight

Mode: setup, analysis

Merges legacy LV flow and Tesson-Shell-based ESOE CDPs into the current Tesson Shell-based CDP.

Usage

```
merge_cdp cdp_path [-replace]
```

Description

You use the resulting CDP along with the process_memory_failures command to convert failure log files produced from previously migrated raw STIL or raw SVF ESOE failure log files to bitmap files. Refer to “[Performing Diagnosis From an ATE Test Program in Offline ATE Mode](#)” in the *Tesson SiliconInsight User’s Manual for Tesson Shell* for details.

When importing an LV flow or Tesson-Shell-based CDP into the current Tesson Shell CDP using this command, the tool does not perform CDP verification. Upon successful execution of this command, all tests originating from the merged CDP become accessible to Tesson SiliconInsight for Tesson Shell.

Arguments

- *cdp_path*

A required string that specifies the path to the LV flow or Tesson Shell-based CDP that you want to merge.

- -replace

An optional switch that causes the tool to replace the existing tests in the current Tesson Shell-based CDP when there are name conflicts with the test names in the CDP that you are merging.

Examples

As shown in the following example, specify the merge_cdp command after specifying the setup specification and invoking Tesson SiliconInsight.

```
set_context patterns -silicon_insight
set_silicon_insight_option -cdp_verification off
set_silicon_insight_option -ignore_uncontacted_pin_data on
read_config_data si.setup
start_silicon_insight
merge_cdp my_old_cdp
```

migrate_layout

Context: patterns -scan_diagnosis

Mode: analysis

Generates an LDB directory hierarchy that contains the specified preexisting LDB.

Usage

```
migrate_layout ldb_directory -ladb preexisting_ldb [-vdb preexisting_vdb]
```

Description

Generates an LDB directory hierarchy that contains the specified preexisting LDB.

Use this command to copy a preexisting LDB into the LDB directory hierarchy that became available with the Tesson Diagnosis 2013.2 release. This command allows you to upgrade to the LDB directory without regenerating the LDB from the LEF/DEF files. The tool automatically copies the preexisting LDB into the new LDB directory.

For the DFM diagnosis flow, use this command to additionally copy a preexisting violations database (VDB), which was generated automatically during DFM diagnosis, into the new LDB directory.

Arguments

- ***ldb_directory***
A required string that specifies the pathname to a new LDB directory.
- ***-ladb* *preexisting_ldb***
A required switch and string pair that specifies the pathname to a preexisting LDB.
- ***-vdb* *preexisting_vdb***
An optional switch and string pair that specifies the pathname to a preexisting VDB.

Examples

The following example generates a LDB directory named mylayoutdb. It then copies the example2.ldb and example_dfm.vdb databases into the LDB directory.

```
migrate_layout newlayoutdb -ladb preexisting.ldb -vdb preexisting_dfm.vdb
```

move_config_element

Context: dft

Mode: analysis

Moves a configuration element from one location to another.

Usage

```
move_config_element object_spec
    [-in_wrapper wrapper_name_or_object]
    [-before before] [-after after] | -first | -last]
    [-replace]
```

Description

Moves a configuration element from one location to another.

Repeatable and non-repeatable wrappers and repeatable properties can be moved from one wrapper to another. Single wrappers and properties can only be moved within their parent wrapper.

Arguments

- *object_spec*

A value that specifies the name of a configuration element or a collection containing one configuration element to be moved. When using the -in_wrapper option, the name is the relative pathname with respect to the specified wrapper. When not using the -in_wrapper option, the name is the complete pathname with respect to the root of the partition. To move an element from one wrapper to the next, you must use the [get_config_elements](#) or the “[get_config_value -object](#)” command to get the object and use it as the object_spec. The moved element must not already exist in the destination wrapper unless the -replace option is used, in which case the existing element will be deleted before being replaced by the moved one.

When the allowed content of the destination parent wrapper is defined with [Wrapper](#), the metadata definition of the source and destination element must be identical (both definition must reference the same metadata definition). If this is not the case, an error message is generated.

- -in_wrapper *wrapper_name_or_object*

An option and value pair that specifies that the configuration element is to be moved inside a specific wrapper. The wrapper_object_spec can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by [get_config_elements](#) or “[get_config_value -object](#)”. When the -in_wrapper is not specified, the destination parent wrapper is assumed to be the same as the source parent wrapper unless the -before or -after option is specified.

- **-before** *before*

An optional switch and value pair that specifies the configuration element in front of which to move the new configuration element. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

When config_object_spec is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. If the -in_wrapper option is not specified, it is a name relative to the root of the partition.

The -before option is mutually exclusive with the -after, -first, and -last options. When none of those four option is specified, -last is assumed.

- **-after** *after*

An optional switch and value pair that specifies a configuration element after which to move the new configuration element. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

When config_object_spec is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. If the -in_wrapper option is not specified, it is a name relative to the root of the partition.

The -after option is mutually exclusive to the -before, -first and -last options. When none of those four option is specified, -last is assumed.

- **-first**

An optional switch that specifies to move the new configuration element before any other element in the parent wrapper.

The -first option is mutually exclusive to the -before, -after and -last options. When none of these four options is specified, -last is assumed.

- **-last**

An optional switch that specifies to move the new configuration element after all other elements in the parent wrapper.

The -last option is mutually exclusive to the -before, -after, and -first options. When none of those four option is specified, -last is assumed.

- **-replace**

An optional switch that suppresses the error that is normally generated when moving a configuration element from one wrapper to another and the leaf_name conflicts with an existing element in the destination wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

The following example moves configuration elements within and across parent wrappers.

```
set_transcript_style input_only
set dft [add_config_element DftSpecification(modal,rtl)]
{/DftSpecification(modal,rtl)}

set iitag [add_config_element IitagNetwork -in_wrapper $dft]
{/DftSpecification(modal,rtl)/IitagNetwork}

set host [add_config_element HostScanInterface(1) -in_wrapper $itiag]
{/DftSpecification(modal,rtl)/IitagNetwork/HostScanInterface(1) }

set sib1 [add_config_element sib(1) -in_wrapper $host]
{/DftSpecification(modal,rtl)/IitagNetwork/HostScanInterface(1)/Sib(1) }

set sib2 [add_config_element sib(2) -in_wrapper $host]
{/DftSpecification(modal,rtl)/IitagNetwork/HostScanInterface(1)/Sib(2) }

set sib3 [add_config_element sib(3) -in_wrapper $host]
{/DftSpecification(modal,rtl)/IitagNetwork/HostScanInterface(1)/Sib(3) }

set sib4 [add_config_element sib(4) -in_wrapper $host]
{/DftSpecification(modal,rtl)/IitagNetwork/HostScanInterface(1)/Sib(4) }

report_config_data $dft
DftSpecification(modal,rtl) {
    IitagNetwork {
        HostScanInterface(1) {
            Sib(1) {
            }
            Sib(2) {
            }
            Sib(3) {
            }
            Sib(4) {
            }
        }
    }
}
move_config_element $sib3 -in_wrapper $sib2
report_config_data $dftDftSpecification(modal,rtl)
> report_config_data $dftDftSpecification(modal,rtl)
```

```
{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(1) {
      }
      Sib(2) {
        Sib(3) {
        }
      }
      Sib(4) {
      }
    }
  }
}

#Because -in_wrapper is not specified, $sib one is moved within its parent wrapper
move_config_element $sib1 -last
report_config_data $dftDftSpecification(modal,rtl)

{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(2) {
        Sib(3) {
        }
      }
      Sib(4) {
      }
      Sib(1) {
      }
    }
  }
}

#example showing the use of a config element name in the -in_wrapper option
move_config_element $sib1 -in_wrapper \DftSpecification(modal,rtl)/
IjtagNetwork/HostScanInterface(1)/Sib(2)

report_config_data $dftDftSpecification(modal,rtl)

{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(2) {
        Sib(3) {
        }
        Sib(1) {
        }
      }
      Sib(4) {
      }
    }
  }
}
```

```
#Because -in_wrapper is not specified, $sib one is moved within its parent wrapper
move_config_element $sib1 -first
report_config_data $dftDftSpecification(modA,rtl)

{
    IjtagNetwork {
        HostScanInterface(1) {
            Sib(2) {
                Sib(1) {
                }
                Sib(3) {
                }
            }
            Sib(4) {
            }
        }
    }
}
```

Related Topics

[add_config_element](#)

move_connections

Context: all contexts

Mode: insertion

Moves connections from pin or net objects to pin objects.

Usage

```
move_connections -from obj_spec1 -to obj_spec2 [-net_name net_name] [-silent]
```

Description

Moves connections from pin or net objects to pin objects.

The **-from *obj_spec1*** argument is assumed to be a net when the **-to *obj_spec2*** argument is one level below the hierarchy of ***obj_spec1***. For example, the following command moves the u1/a pin to the *net* by the name of u1/a, instead of the pin, because u1/u2/b is below u1.

move_connections -from u1/a -to u1/u2/b

In another example, u2/b is not below u1. In this case, the command creates the connection to the u2/b *pin* instead of to the net.

move_connections -from u1/a -to u2/b

The specific behavior of the **move_connections** command varies based on the object type found inside ***obj_spec1***.

If the object within ***obj_spec1*** is:

- A pin

The net connected to this pin is moved to the pin specified in ***obj_spec2*** and the pin in ***obj_spec1*** is left unconnected.

- A net that is also an input port

The net is renamed everywhere it is used and is then connected to the pin in ***obj_spec2***. The input port is left with no fanout. The renamed net name is the specified **-net_name** value or the original net name. When you do not specify the **-net_name** option, the tool uniquifies the original name using the **_ts#** suffix.

- A net that is also an output port

The net is renamed everywhere it is used and is then connected to the pin in ***obj_spec2***. The output port is left with no fanin. The renamed net name is the specified **-net_name** value or the original net name. When you do not specify the **-net_name** option, the tool uniquifies the original name using the **_ts#** suffix.

- A net that is not also a port

The net is renamed everywhere it is used, except at its driver source, and is then connected to the pin in *obj_spec2*. The driver source is left connected to the original net with no fanout. The renamed net name is the specified -net_name value or the original net name. When you do not specify the -net_name option, the tool uniques the original name using the _ts# suffix by default. You can change the default with “set_insertion_options -net_uniquification_suffix”.

The *obj_spec1* must be a Tcl list of one or more pin or net names, or a collection of one or more pin or net objects. The *obj_spec2* must be a Tcl list of one or more pin names or a collection of one or more pin objects. The *obj_spec1* and *obj_spec2* must have exactly the same size.

Note

 The -from and -to objects can exist at different levels of hierarchy, but a move across hierarchy is not recommended because it is possible to create a lot of unnecessary ports through the hierarchy. You should limit moves to within a single hierarchy by instantiating the intercepting cell in the parent instance of the node to be intercepted.

This command returns a collection containing the net object that ends up being connected to the pin object in *obj_spec2*. If the -from object is a pin, the command returns the nets that were moved. If the -from object is a net, the command returns the nets that were created.

If an object does not exist in *obj_spec1* or *obj_spec2* and the -silent option is specified, the object is ignored and the move action skipped. If an object does not exist and the -silent option is not specified, the tool generates an error.

You must be careful when using the move_connections command inside a non-uniquified design. You must only perform the move or disconnection once per module and once per generate loop count. For information about how to perform this robustly, see how the parent_instance and the leaf_name_hash attributes are used in Example 5 in the [get_dft_cell](#) command description. For information about the leaf_name_hash attribute, refer to the its description in “[Instance](#)” on page 3069.

Arguments

- **-from *obj_spec1***

A required switch and value pair that specify the object to be moved. The *obj_spec1* must be a Tcl list of one or more pin or net names, or a collection of one or more pin or net objects.

- **-to *obj_spec2***

A required switch and value pair that specifies the pin to which the net is moved to. The *obj_spec2* must be a Tcl list of one or more pin names, or a collection of one or more pin objects. The *obj_spec1* and *obj_spec2* must have exactly the same size.

- **-net_name *name***

An optional argument that specifies the base net name to create when moving a net. If you do not use this switch, the tool uses the original net name uniques using the _ts# suffix.

- **-silent**

An optional argument that ignores specified objects that do not exist and aborts the move_connections execution without generating an error message. If the move_connections command does not execute successfully, the returned net object is null.

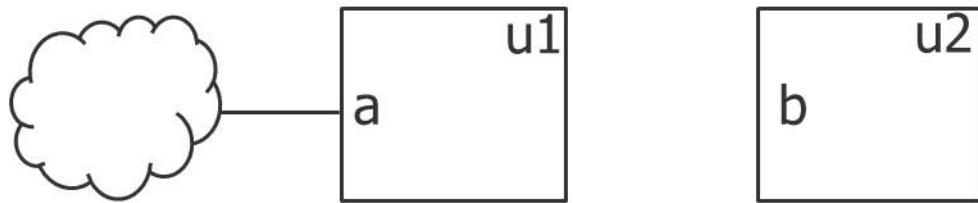
Examples

Example 1

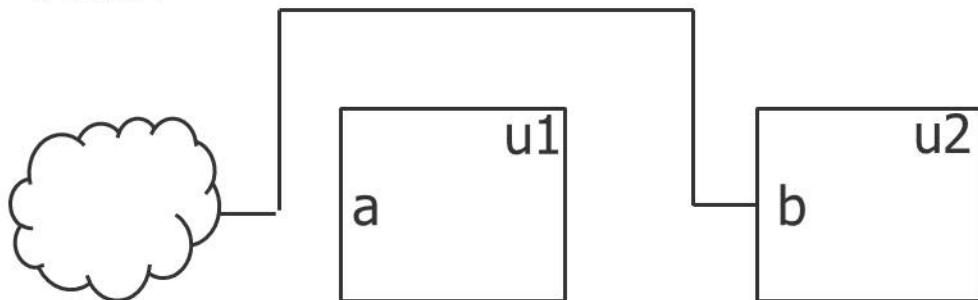
This following example moves the net connected to input pin u1/a to the input pin u2/b. After the operation, pin u1/a is left with no driver.

move_connections -from u1/a -to u2/b

Before:



After:



Example 2

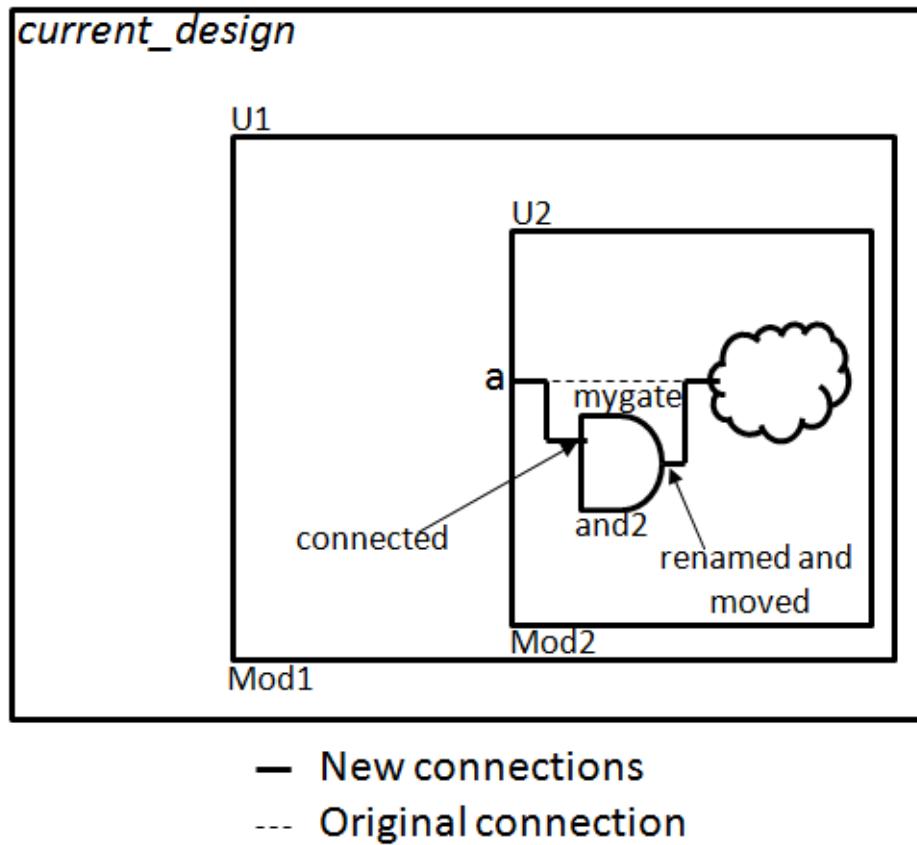
The following example shows how to intercept an input pin the way it is done when you use the **intercept_connection** command. By using the **get_nets** introspection command, the net named u1/u2/a is returned instead of the pin as in the previous example. The parent_instance of the net automatically returns u1/u2 which results in the intercepting gate being placed inside the module as required.

```

set net [get_net u1/u2/a]
set gate_inst_name [lindex [get_attribute_value_list $net -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $net -to [get_pins y -of_instance $gate_inst]
create_connections $net [get_pins a -of_instance $gate_inst]

```

Figure 4-7. Intercepting an Input Pin From Inside An Instance

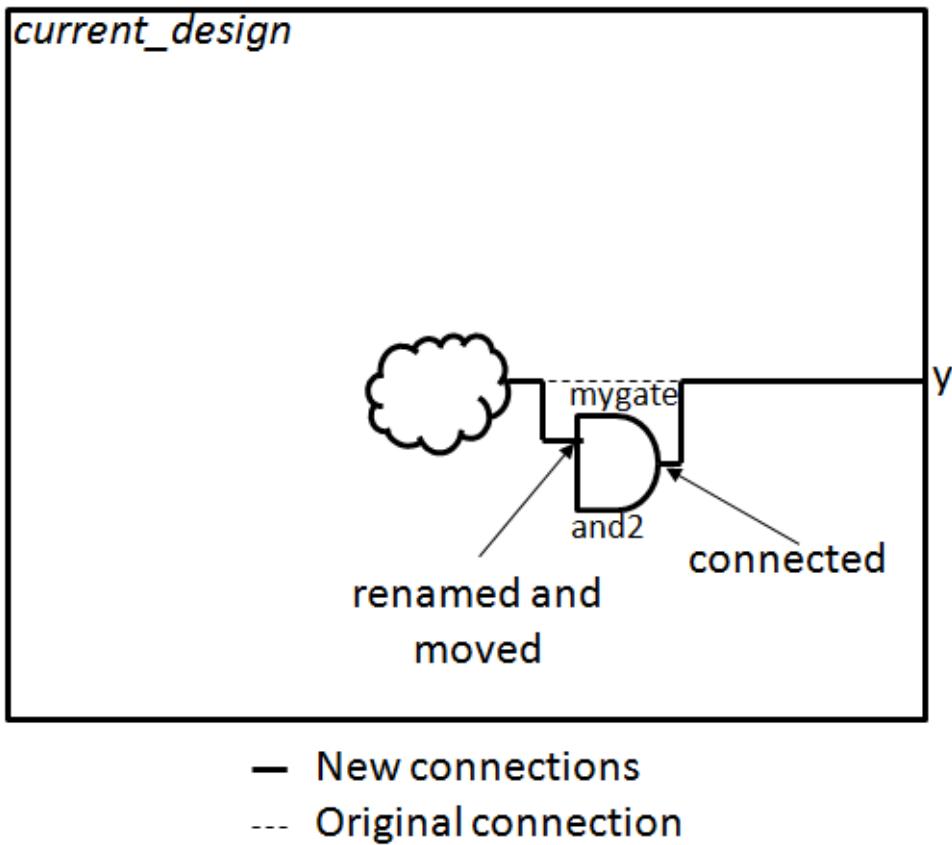


Example 3

This following example shows how to intercept an output port from the inside of the current design. The parent_instance of the port automatically returns a null value, which results in the intercepting gate being placed inside the root module as required. The net names created by the move_connections and create_connections commands are derived from the net named being moved or connected. You can use the -net_name option if you want a specific net name to be used instead.

```
set port y
set gate_inst_name [lindex [get_attribute_value_list $port -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod mygate -allow_inst_uniquification]
move_connections -from $port -to [get_pins a -of_instance $gate_inst]
create_connections $port [get_pins y -of_instance $gate_inst]
```

Figure 4-8. Intercepting an Output Port From Inside the Current Design

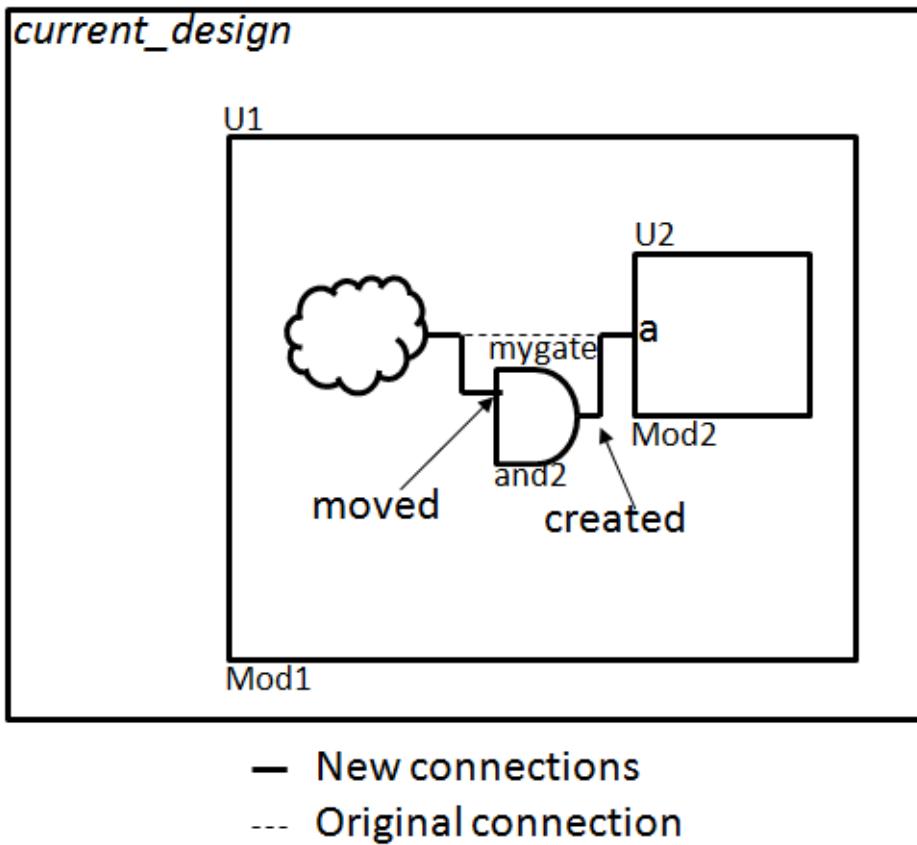


Example 4

This following example shows how to intercept an input pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new unqualified name. The net that was connected to the pin u1/u2/a ends up connected to pin "a" of \$gate_inst. Following the move_connections operation, pin u1/u2/a is left open. The create_connections command is then used to reconnect it to the output of the intercepting gate.

```
set pin u1/u2/a
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins a -of_instance $gate_inst]
create_connections $pin [get_pins y -of_instance $gate_inst]
```

Figure 4-9. Intercepting an Input Pin From Outside An Instance

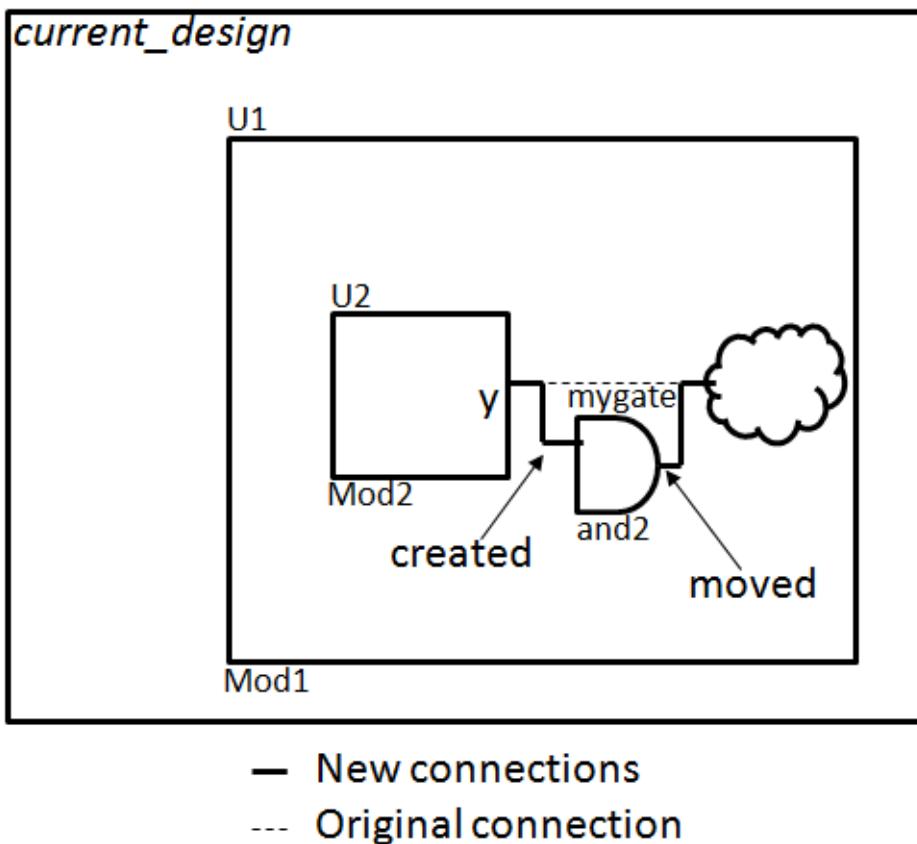


Example 5

The following example shows how to intercept an output pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new unqualified name. The net that was connected to pin u1/u2/y ends up connected to pin y of \$gate_inst. Following the move_connections, pin u1/u2/y is left open. The create_connections command is then used to reconnect it to the input of the intercepting gate.

```
set pin u1/u2/y
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins y -of_instance $gate_inst]
create_connections $pin [get_pins a -of_instance $gate_inst]
```

Figure 4-10. Intercepting an Output Pin From Outside An Instance



Related Topics

[create_connections](#)
[delete_connections](#)
[intercept_connection](#)

no_transcript

Context: unspecified, all contexts

Mode: all modes

Turns off transcripting for the specified commands.

Usage

`no_transcript { command1 command2 ... }`

Description

Turns off transcripting for the specified commands.

This command is useful when you are writing Tcl procedures in which you do not want your embedded commands to show up in the transcript or log file. The no_transcript command is also not transcripted.

Arguments

- ***command1 command2 ...***

A required string that specifies the commands that you do not want to appear in the transcript or log file.

Examples

The following excerpt from a Tcl procedure (see [get_dft_cell](#), Example 5), shows a usage example for the no_transcript command:

```
#####
#### Section 5: Retrieves leaf instance name of gate if node is a
#### repeated module instance and skips the interception
####

if {[info exists insertion_parent_leaf_instance_array($key)] && \
    [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
    set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
    if {$leaf_block_id ne ""} {
        set cell_inst [get_instance \
            "${parent_instance_name}/${leaf_block_id}.${leaf_cell_inst}"]
    } else {
        set cell_inst [get_instance \
            "${parent_instance_name}/${leaf_cell_inst}"]
    }
} else {
    if {$leaf_block_id ne ""} {
        set cell_inst_name \
            "${parent_instance_name}/${leaf_block_id}.insertion_${cell_sel}"
    } else {
        set cell_inst_name \
            "${parent_instance_name}/insertion_${cell_sel}"
    }
    no_transcript {
        set cell_inst [create_instance $cell_inst_name -of_module \
            $dft_cell -allow_instance_uniquification]
    }

    set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
        leaf_name] 0]
    set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
        -name leaf_block_id] 0]
    if {$leaf_block_id ne ""} {
        # Remove the block_id part from the leaf instance name
        set leaf_name [string range $leaf_name \
            [string length $leaf_block_id] end]
    }
    set insertion_parent_leaf_instance_array($key) $leaf_name

    if {$insertion_type == "sink"} {
        no_transcript {
            move_connections -from $node_object \
                -to [get_pins ${input0} -of_instances $cell_inst]
            create_connections [get_pins ${output} -of_instances \
                $cell_inst] $node_object
        }
    } else {
        no_transcript {
            move_connections -from $node_object -to [get_pins ${output} \
                -of_instances $cell_inst]
            create_connections $node_object [get_pins ${input0} \
                -of_instances $cell_inst]
```

```
        }
    }

    if {$connect_in2} {
        no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
                $cell_inst]
        }
    }

    if {$connect_sel} {
        no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
                $cell_inst]
        }
    }
    return $cell_inst
}
```

Related Topics

[echo](#)

[get_transcript_style](#)

[set_logfile_handling](#)

[set_transcript_style](#)

[open_layout](#)

Context: patterns -scan_diagnosis

Mode: analysis

Opens an existing Tesson Diagnosis tool-compatible Layout Database directory.

Usage

Layout-Aware Diagnosis Flow

```
open_layout layout_database_name [-VERIFY [-EXTRA_LAYOUT_hierarchy  
extra_levels_of_hierarchy_in_layout |  
-EXTRA_DESign_hierarchy extra_levels_of_hierarchy_in_design>] ]  
[-THreshold percentage_match] [-chip_design_name top_design_name]  
[-do_tolerant_match {on | off}]
```

SVDB Layout Marker Flow

```
open_layout -SVDB svdb_directory_name [-32]
```

Description

Layout-Aware Diagnosis Flow

Opens an existing Tesson Diagnosis tool-compatible Layout Database directory.

Any subsequent usage of this command first closes any existing layout before Tesson Diagnosis opens a new Layout Database. See “[Layout-Aware Diagnosis](#)” in the *Tesson Diagnosis User’s Manual* for complete information on this flow.

In hierarchical layout-aware diagnosis, you must use the -chip_design_name option when you have a core that is instantiated in more than one design. See “[Diagnosis for Hierarchical Design](#)” in the *Tesson Diagnosis User’s Manual* for more information.

Layout-Aware Diagnosis Flow with DFM

Opens an existing Tesson Diagnosis tool-compatible Layout Database and if this database contains imported DFM information, the layout-aware diagnosis results will be annotated with DFM hit results.

For more information, see “[Diagnosis for Design for Manufacturability Analysis](#)” in the *Tesson Diagnosis User’s Manual*.

SVDB Layout Marker Flow

Checks out a *calibre* license and invokes the 64-bit Calibre Query Server software in the background on workstations supporting 64-bit. Otherwise, Tesson Diagnosis executes this command in 32-bit mode. Before using this command, you must have a valid SVDB directory containing net name information and set the CALIBRE_HOME environment variable to your Calibre software tree.

Arguments

- ***layout_database_name***

A required string that specifies the name of the existing Layout Database directory created with the [create_layout](#) command. For hierarchical layout-aware diagnosis, the LDB must be an instance-aware core-level LDB.

You can use the tab completion method to list possible files for ***layout_database_name*** that exist in the current directory. For example, specifying “open_layout mem” and pressing the Tab key will list the files with names starting with mem in the current directory.

Layout-Aware Diagnosis Flow

- **-VERIfy**

An optional switch that specifies to perform verification if the current design has not been validated against the current Layout Database. Using this option ensures that in subsequent runs the layout database can be opened for the current design without having to re-run verification.

At the time of opening the layout, Tessonnt Diagnosis checks whether the current design has been verified against the current Layout Database. If the current design has not been validated, it performs verification. During verification, layout rule violation information is added to the layout database.

For each flat model used during diagnosis, you must first perform the verification step. You can do this with the `open_layout -verify` command. With subsequent diagnosis runs using a verified flat model/LDB combination, you can skip the verification step by just specifying `open_layout`. If you need to verify many flat models against the same LDB, you can use `open_layout -verify` to verify them in parallel in separate diagnosis sessions.

Layout verification rules can be divided into two broad categories: those that check the consistency of LEF/DEF files themselves, and those that match design information with the layout. Only the second type of layout verification rules are re-evaluated when a layout database is verified against a new design. Because the LEF/DEF layout information has not changed, the first type of rules do not need to be re-run.

- **-EXTRA_LAYout_hierarchy *extra_levels_of_hierarchy_in_layout***

An optional switch and string pair that removes extra levels of layout hierarchy such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

You use this switch if your design is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

This switch can only be used in conjunction with the `-verify` switch.

- **-EXTRA_DESign_hierarchy *extra_levels_of_hierarchy_in_design***

An optional switch and string pair removes extra levels of design hierarchy such that the layout hierarchy and design hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

You use this switch if your layout is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

This switch can only be used in conjunction with the `-verify` switch.

- **-THreshold *percentage_match***

An optional switch and integer pair specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent verification of the layout database. The default threshold value is 85 percent.

- **-chip_design_name *top_design_name***

An optional switch that specifies a top-level design whose core instance information is included in the specified instance-aware core-level LDB. This option only pertains to hierarchical layout-aware diagnosis and is required when you have a core that is instantiated in more than one design.

A chip-mapped core-level LDB can store information on instances of this core in multiple designs. This switch specifies which top-level design in which the core level LDB is included when performing hierarchical layout-aware diagnosis. This allows Tesson Diagnosis to report the correct chip-level coordinates for suspects reported in core-level diagnosis. If this switch is not used, the tool only reports core-level coordinates for suspects reported in the core-level diagnosis.

- **-do_tolerant_match {on | off}**

An optional switch and literal that specifies whether to allow escape-tolerant name matching during layout verification when you specify `open_layout -verify`. By default, the tool tolerates differences in escapes in instance path names and net path names. This switch only applies to name matching during verification and has no impact to names stored in the LDB. The names stored in the LDB are the original LEF/DEF names.

SVDB Layout Marker Flow

- **-SVDB *svdb_directory_name***

A required switch and string pair that specifies the name of a valid SVDB directory. The SVDB must contain net name information.

- **-32**

An optional switch invoking the 32-bit version of the Calibre Query Server software. The 64-bit version is the default.

Examples

Example 1

In the following example, Tesson Diagnosis loads the existing Layout Database (*design.layout*) into memory:

open_layout *design.layout*

Example 2

The following example illustrates using the -VERIfy switch with the open_layout command for two designs: *DesignA.flat* and *DesignB.flat*. Except for different pin constraints, the two designs are essentially identical. Because a high percentage of the logic overlaps, both designs use the same Layout Database for layout-aware diagnosis.

To perform layout-aware diagnosis on these two designs using the same Layout Database, you do the following:

1. Create and verify the Layout Database, in this example *DesignA_layout_database*, with *DesignA.flat* using the [create_layout](#) command.
2. Invoke Tesson Diagnosis in scan diagnosis mode and read in the *DesignB.flat* flat model with the [read_flat_model](#) command.
3. Using the open_layout command and -VERIFY switch, open the *DesignA_layout_database* with the *DesignB.flat* model as in the following:

open_layout -verify DesignA_layout_database

This verifies *DesignB.flat* against the newly-created *DesignA_layout_database*.

4. Proceed with layout-aware diagnosis with *DesignB.flat*.

Related Topics

[close_layout](#)

[create_layout](#)

[delete_layout_verification](#)

[report_layout_rules](#)

open_pattern_set

Context: patterns -ijtag

Mode: analysis

Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.

Usage

```
open_pattern_set pattern_set_name [-timeplate timeplate | -tester_period tester_period]  
      [-tck_ratio tck_ratio] [-no_initial_ireset] [-explicit_ireset] [-ireset_in_test_setup_proc ]  
      [-ignore_compliance_values] [-replace]
```

Description

Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.

A populated pattern set is closed using the [close_pattern_set](#) command. An existing pattern set cannot be overwritten unless it is first deleted or the -replace option is used.

PDL commands can only be specified when a pattern set is open with the exception of the iReset and iCall commands which can also be specified within the test_setup and test_end procedures in the patterns -scan context.

You use the -timeplate switch to specify the timeplate for this pattern set. Specifying a different timeplate is rarely needed as the default timeplate is adequate for standard IJTAG operations. Tessent Shell automatically defines the appropriate timeplate based on the specified off value for TCK and the specified -tck_ratio option value. The only time you need a different timeplate is in the rare situation you want to define clock waveforms with more than two edges.

The -tck_ratio option defaults to 1. You must specify a larger value if you have synchronous clocks ([add_clocks](#) commands without the -period option). In this case, you specify -tester_period to match your system clock requirements and specify a -tck_ratio value such that TCK toggles at a rate that is not higher than it was designed for.

If all your system clocks are asynchronous ([add_clocks](#) commands with the -period option), allow -tck_ratio to default to 1 and specify -tester_period to match your TCK period requirements. [Figure 4-11](#) through [Figure 4-18](#) illustrate the TCK timing and the position of the ScanOut strobing relative to the rising edge of TCK. Notice how, independent of the off value for TCK and the specified tck_ratio value, ScanOut is always strobed one percent of the TCK period before the rising edge of TCK.

With a custom timeplate, you can specify different timing for the measurements and forces of ScanIn, ScanOut and TCK; however, the measurement time for ScanOut must be before the rising of TCK and after the falling edge of TCK when the off value for TCK is 1. For more

information about timeplates, refer to “[Timeplate Definition](#)” in the *Tessent Shell User’s Manual*.

Figure 4-11. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 0

```
timing_variables =
  tester_period = 10;
  strobe_1 = (0.24 * tester_period);
  t_time = (0.25 * tester_period);
  t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
  force_pi 0;
  measure_po strobe_1;
  pulse_clock t_time t_width;
  period tester_period;
end;
```

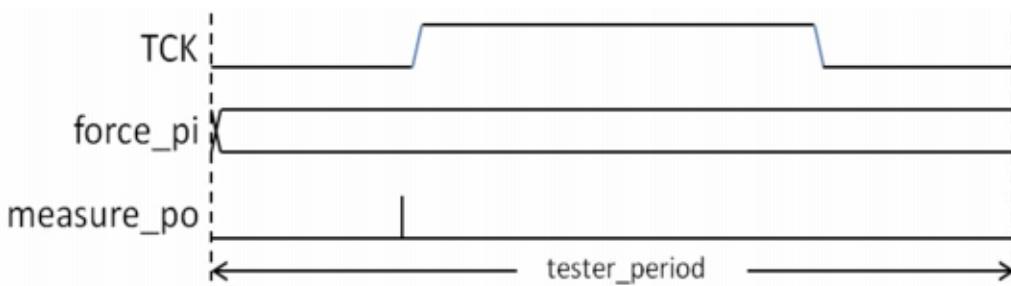
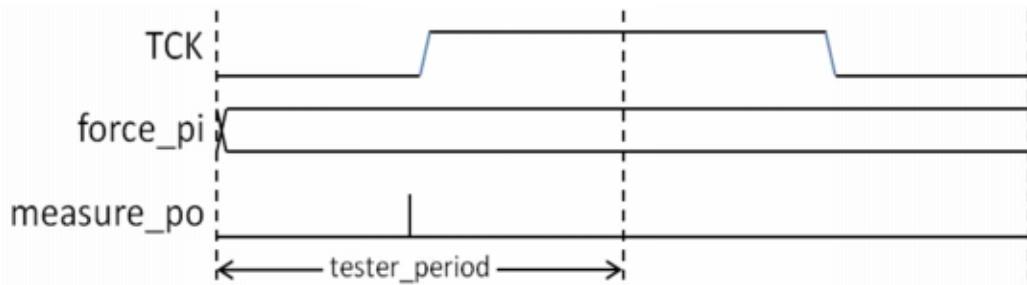


Figure 4-12. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 0

```
timing_variables =
    tester_period = 10;
    strobe_1 = (0.48 * tester_period);
    f_time = (0.5 * tester_period);
    t_time = (0.25 * tester_period);
    t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
    force_pi 0;
    measure_po strobe_1;
    force TCK f_time;
    pulse_clock t_time t_width;
    period tester_period;
end;
```

**Figure 4-13. Scan Frame Timing with tck_ratio = 4 and TCK Off Value of 0**

```
timing_variables =
    tester_period = 10;
    strobe_1 = (0.96 * tester_period);
    t_time = (0.25 * tester_period);
    t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
    force_pi 0;
    measure_po strobe_1;
    force TCK 0;
    pulse_clock t_time t_width;
    period tester_period;
end;
```

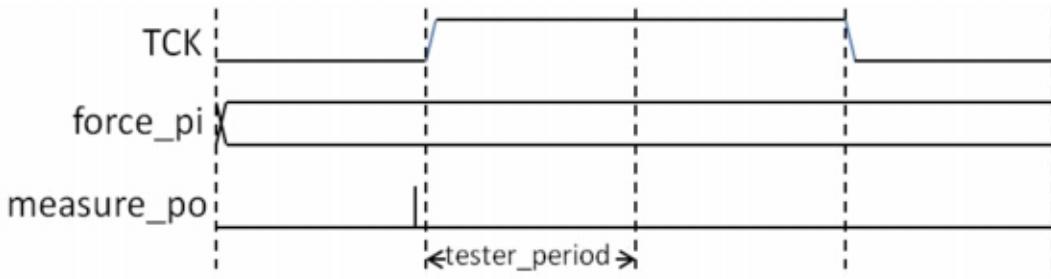


Figure 4-14. Scan Frame Timing with tck_ratio >= 8 and TCK Off Value of 0

```
timing_variables =
  tester_period = 10;
  strobe_1 = (0.96 * tester_period);
  t_time = (0.25 * tester_period);
  t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
  force_pi 0;
  measure_po strobe_1;
  force tck 0;
  pulse_clock t_time t_width;
  period tester_period;
end;
```

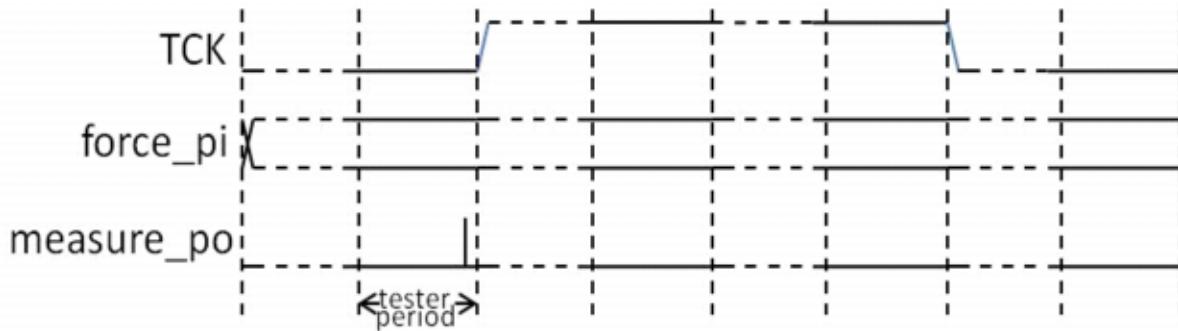


Figure 4-15. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 1

```
timing_variables =
  tester_period = 10;
  strobe_1 = (0.74 * tester_period);
  t_time = (0.25 * tester_period);
  t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
  force_pi 0;
  measure_po strobe_1;
  pulse_clock t_time t_width;
```

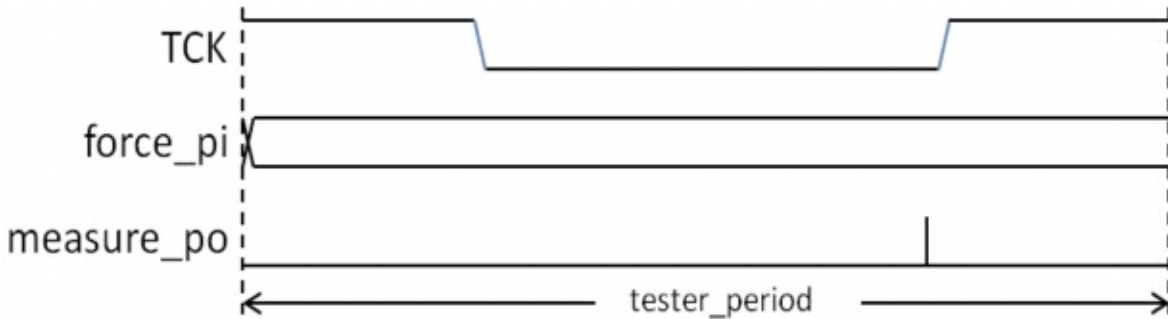
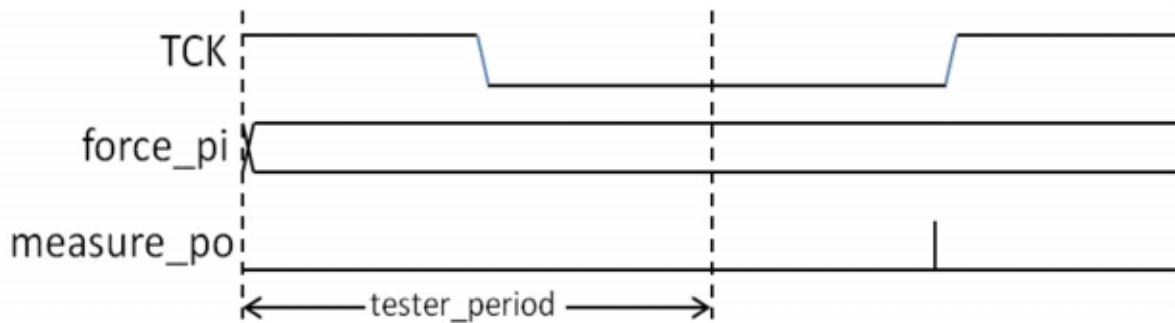


Figure 4-16. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 1

```
timing_variables =
    tester_period = 10;
    strobe_1 = (0.48 * tester_period);
    f_time = (0.5 * tester_period)
    t_time = (0.25 * tester_period);
    t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
    force_pi 0 ;
    measure_po strobe_1;
    force TCK f_time;
    pulse_clock t_time t_width;
```

**Figure 4-17. Scan Frame timing with tck_ratio = 4 and TCK Off Value of 1**

```
timing_variables =
    tester_period = 10;
    strobe_1 = (0.96 * tester_period);
    t_time = (0.25 * tester_period);
    t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
    force_pi 0;
    measure_po strobe_1;
    force TCK 0;
    pulse_clock t_time t_width;
    period tester_period;
end;
```

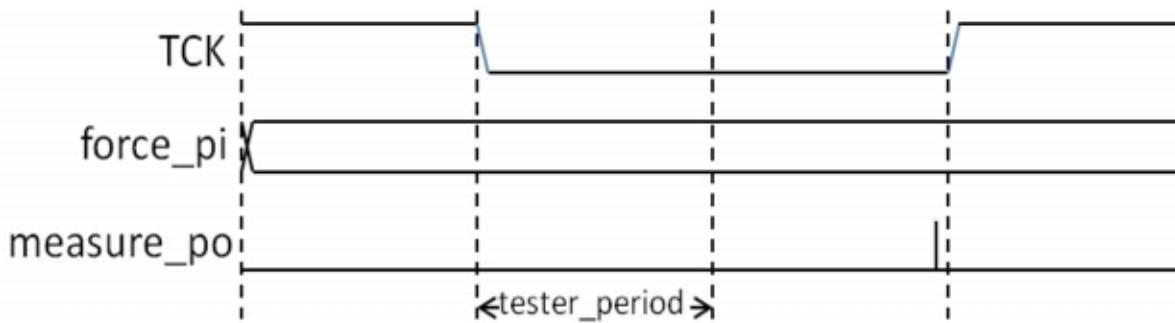


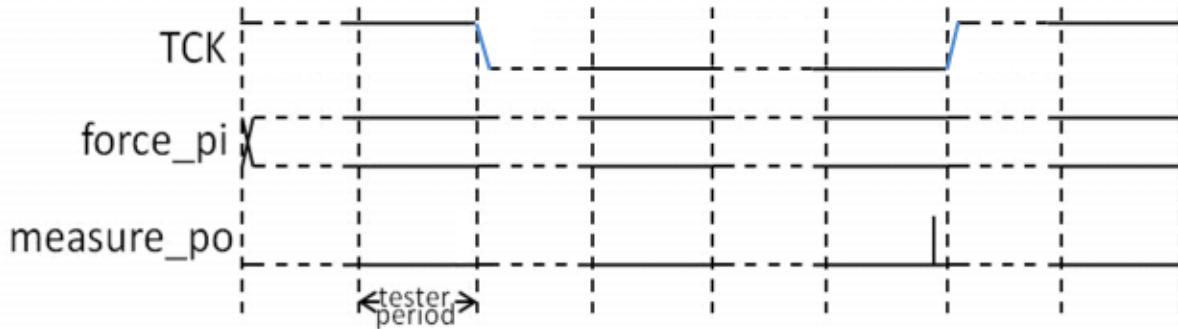
Figure 4-18. Scan Frame timing with tck_ratio >= 8 and TCK Off Value of 1

```

timing_variables =
  tester_period = 10;
  strobe_1 = (0.96 * tester_period);
  t_time = (0.25 * tester_period);
  t_width = (0.5 * tester_period);
end;

timeplate tesson_ijtag =
  force_pi 0;
  measure_po strobe_1;
  force tck 0;
  pulse_clock t_time t_width;
  period tester_period;
end;

```



Arguments

- *pattern_set_name*

A required string that defines the name of the pattern set. This name uniquely identifies the pattern set for later use with `write_patterns`. This name also specifies the name of the iProc, if the pattern set is written out in -pdl format.

Note

 The pattern set names “test_setup” and “test_end” are not allowed.

- -timeplate *timeplate*

An optional switch and string pair that specifies the name of a valid timeplate, defined in a procedure file reference with the `set_procfile` command. This option is mutually exclusive with the `-tester_period` option as the `tester_period` is specified within the timeplate when the `-timeplate` option is used.

- -tester_period *tester_period*

An optional switch and value pair that specifies the tester period when the built-in timeplate, shown in [Figure 4-11](#) through [Figure 4-18](#), is used. When neither `-timeplate` nor `-tester_period` is used, Tessent Shell uses the first timeplate that can be found and that has appropriate timing specifications for the TCKPorts and the ScanInterface control ports. This could be, for example, a timeplate in a procfile imported by means of the [Command](#)

[Dictionary \(S - Z\)](#) `set_procfile_name` command or the built-in timeplate from a previous `open_pattern_set` command. If there is no suitable timeplate, a new one with a tester period of 100ns is created.

- `-no_initial_ireset`

An optional switch that specifies to create a pattern set that is to run after another one while keeping power and clocks active between them. You may need to create a pattern set of this type if you need to insert some ATE actions in the middle of a test such as changing a voltage. These types of pattern sets are exported together when writing out the Verilog test bench and as separate STIL or WGL files when exporting the pattern files to use on the ATE. You should use special care when running these patterns on the ATE because the lack of a reset at the beginning of the pattern makes it impossible to quit in the middle of the pattern and resume from the start when the capture memory is filled up. Data logging on such patterns can only be done in one pass and to the extent of the size of the capture memory on the ATE equipment.

The `-no_initial_ireset` option cannot be used on the first `open_pattern_set` command of a session. When used on a subsequent `open_pattern_set` command, the end ICL state of the previous `pattern_set` is used as the beginning ICL state for the current pattern set.

- `-explicit_ireset`

An optional switch that allows to delay the first iReset of a pattern set until the first action that requires the IJTAG network to be in a defined state. In some situations, it is desirable to apply certain actions to the IJTAG network before the first iReset happens, for example, by means of the commands [iForcePort](#) and [iComparePort](#). The `-explicit_ireset` switch suppresses the implied iReset that otherwise would be added to the beginning of the pattern set by the `open_pattern_set` command. Prior to the first explicit usage of the command `iReset` in the pattern set, the `-explicit_ireset` switch prevents the usage of all commands that require the IJTAG network to be in a defined state, which are [iRead](#), [iWrite](#), [iRunLoop](#), [iClock](#), [iClockOverride](#), [iMerge](#), [iTakes](#), [iRelease](#), [create_icl_verification_patterns](#) and [import_patterns_from_svf](#). Other commands, like `iForcePort` and `iComparePort`, can be used before the first `iReset`.

- `-ireset_in_test_setup_proc`

An optional switch that instructs the IJTAG retargeter to assume that the `test_setup` procedure contains the appropriate commands to reset the IJTAG network. When this switch is used, subsequent PDL retargeting assumes that the registers with a `ResetValue` are in their reset state, the SIBs are in their initial state and so on. You are responsible for the reset of the IJTAG network in the `test_setup` procedure. If there is a TAP controller, you are also responsible for the required transition from the `RESET` state to the `IDLE` state in the `test_setup` procedure. The `-ireset_in_test_setup_proc` switch can only be used if there is a `test_setup` procedure in the procfile. The `-ireset_in_test_setup_proc` and `-no_initial_ireset` switches are mutually exclusive.

- `-tck_ratio tck_ratio`

An optional switch and value pair that specifies the ratio between the required TCK period and the `tester_period` used for this pattern set. The `tester period` is the value specified with

the -tester_period option when using the built-in timeplate or the period entry of the timeplate referenced by the -timeplate option. The default value is 1.

- **-ignore_compliance_values**

An optional switch that instructs the tool to ignore the compliance_value attributes of the ICL ports of the top module. When this switch is omitted, the retargeter applies the compliance values to the ICL ports right at the beginning of the pattern set.

- **-replace**

An optional switch that specifies to replace an existing pattern set with the same name.

Return Values

None

Examples

The following example opens a pattern set called pat1 using the built-in timeplate described in [Figure 4-11](#). The tester_period is over-written to 50 ns. The same pattern set is later replaced by one where the timeplate TP1 is used. TP1 is a timeplate defined inside a procfile and referenced by the set_procfile command in setup mode.

```
open_pattern_set pat1 -tester_period 50ns
iCall myProc
close_pattern_set

open_pattern_set pat1 -timeplate TP1 -replace
```

Related Topics

[close_pattern_set](#)
[get_open_pattern_set](#)
[report_pattern_sets](#)
[reset_open_pattern_set](#)

open_tsdb

Context: all contexts

Mode: all modes

Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.

Usage

open_tsdb *directory_path_list* [-silent]

Description

Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.

Many actions are taken on the opened TSDBs at various steps in the design flow. Some of those actions are described here:

- When the [set_current_design](#) command is invoked, the tool automatically reads in the most recent *module_name.format_interface* file using the [read_verilog](#) or [read_vhdl](#) -interface_only option if the corresponding *module_name* does not already exist in memory. This enables you to only load the modules associated with the current design and have the interface view of all child physical or sub-blocks loaded automatically.
- All [dft_inserted_designs](#) and instrument subdirectories are automatically searched, and ICL files corresponding to the design modules are automatically loaded after design elaboration.
- The *is_physical_module* attribute (see the “Built-In Attributes” table in “[Module](#)” on page 3058) is automatically set on all modules that are found in a TSDB and that were declared as physical blocks using the [set_design_level](#) command when they were processed by the tool.

Arguments

- *directory_path_list*

A required value containing an absolute or relative path to a TSDB directory, or a Tcl list containing one or many absolute or relative paths to TSDB directories. The tool verifies that each specified directory path is an existing TSDB directory and that only one copy of an instrument and [dft_inserted_designs](#) directories exists for any given *module/design_id* combination in all opened TSDBs; if this is not the case and you get this error, you must close the redundant TSDB directories using the [close_tsdb](#) command. See also “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763.

- -silent

An optional switch that suppresses the warning that is normally generated if the specified *directory_path* is already opened.

Examples

The following example opens three TSDB directories with one command invocation.

```
open_tsdb [list ${tsdb_home}/corea.tsdb \
           ${tsdb_home}/coreb.tsdb \
           ${tsdb_home}/corec.tsdb ]get_tsdb_list

/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

[close_tsdb](#)
[get_tsdb_list](#)
[get_tsdb_output_directory](#)
[set_tsdb_output_directory](#)

[open_visualizer](#)

Context: unspecified, all contexts

Mode: all modes

Opens the main DFTVisualizer window.

Usage

`open_visualizer [-Display window | -Restart]`

Description

Opens the main DFTVisualizer window.

You can also use the following commands to open DFTVisualizer and display the Hierarchical Schematic and Console windows. The Design Browser and Console windows open by default:

- `analyze_drcViolation`
- `add_display_instances`
- `analyze_fault` (Tessent FastScan and Tessent TestKompress only)
- `display_specification`

Displayed data remains persistent throughout a tool session.

Arguments

- `-Display window`

An optional switch and literal pair that specifies which windows display when DFTVisualizer opens. The Design Browser and Console windows will open by default if the Display argument is not specified. Valid literals include:

`FLAt_schematic` — opens the Flat Schematic window.

`HIEarchical_schematic` — opens the Hierarchical Schematic window.

`DAta` — opens the Data window.

`Wave` — opens the Wave window.

`SEarch` — opens the Global Search window.

`TExt_editor` — opens the Text Editor window.

`TEST_structures` — opens the Test Structures window.

`Configuration_data` — opens the Configuration Data window.

- `-Restart`

An optional -Restart switch that enables you to restart DFTVisualizer following an application error in DFTVisualizer without restarting the entire tool. This eliminates the need to exit and re-invoke the parent application.

This option closes the current DFTVisualizer window and opens a new DFTVisualizer session displaying either the default windows normally opened at invocation or those windows specified to be open in your saved preferences. This option does not restore any of the tool data contained in DFTVisualizer windows prior to the error.

Tip

If you execute the same set of commands that resulted in the application locking up, the same error will occur and the application will lock up again. You can use the -Restart option and then avoid the same error a second time by following a different set of steps or by skipping the step that resulted in the error.

Examples

Example 1

The following example opens DFTVisualizer and displays the Flat Schematic, Hierarchical Schematic, and Wave windows.

`open_visualizer -display FLA HIE wave`

More information on the invocation and usage of DFTVisualizer can be found in the *Tessent Shell User's Manual*.

Related Topics

[close_visualizer](#)

[write_window_contents](#)

[set_visualizer_preferences](#)

order_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Reorders the internal test pattern set as specified.

Usage

```
order_patterns [number_of_passes | Auto percentage]
  [-Init_order Normal | REverse | Random] [-Simulate [-PATterns_per_pass integer]]
  [{-CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average]
  |-SHift_power LOAD | RESPonse}
  [-ASCending | -DESCending] [-REMOVE_ineffective_patterns ON | OFF}
  {-INSTance instance_name} | {-MODULE module_name}]]
```

Description

Reorders the internal test pattern set as specified.

Test patterns can be rearranged to detect the most faults first, which is useful for:

- Truncating the test pattern set in order to fit it in the ATE memory.
- Detecting failing chips as early as possible during testing.

Multiple load patterns are ranked according to the average number of faults detected per scan load, ensuring that on average the patterns that detect the most faults per scan load are first.

The order_patterns command is not supported when multiple detection is enabled. This command also does not support MacroTest or parametric patterns.

This is a multiprocessing command. For more information about multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

Note

 Using the order_patterns command turns off pattern classification until you issue a “set_pattern_classification On” command.

Arguments

- *number_of_passes*

An optional integer that specifies the number of passes to make through the data when reordering the pattern set. By default, one pass is made.

- *Auto percentage*

An optional switch and integer pair that reorders the pattern set iteratively until further reordering produces negligible improvement in the number of patterns needed to achieve a given fault coverage benchmark. The tool determines the given fault coverage benchmark

by subtracting percentage from the total fault coverage (FC) achieved using all the patterns in the internal pattern set. The percentage argument must be a non-negative number less than or equal to FC.

Reordering stops when either of the following conditions exist:

- The tool completes a maximum of 100 iterations.
- The number of patterns that is needed to achieve the given fault coverage benchmark decreases by less than 1% from one iteration to the next. This stop point can be stated mathematically as:

$$(PFCb \text{ })_{\text{previous_pass}} - (PFCb \text{ })_{\text{current_pass}} < (PFCb \text{ })_{\text{previous_pass}} \times .01$$

where: PFCb = number of patterns needed to achieve the fault coverage benchmark

FCb = FC - percentage = fault coverage benchmark

FC = total fault coverage achieved using all the patterns in the pattern set

- -Init_order Normal | REverse | Random

An optional switch and literal pair that specifies the initial pattern ordering to use as a starting point for pattern reordering. The literal choices are as follows:

Normal — A literal that specifies the order of the current internal pattern set as the initial pattern order.

REverse — A literal that specifies to use the reverse order of the current internal pattern set as the initial pattern order.

Random — A literal that specifies to randomize the order of the current internal pattern set, and use the random order as the initial pattern order.

- -Simulate

An optional switch that fault simulates the test patterns after reordering.

- -PATterns_per_pass *integer*

An optional switch and positive integer pair, used with the -simulate option, that specifies how many patterns to create and simulate in each pass. The integer value must be between 1 and 64. By default, the tool creates and simulates 64 patterns in each pass. When the value is 1, the tool generates and simulates every pattern, and so on.

A smaller value is appropriate if the expected pattern count is small. A typical scenario for using a small integer value is IDDQ pattern ordering.

- -CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average]

Optional switch and literal pair that orders test patterns based on switching activity during capture. Literal options include:

ST_Peak — Peak of state transitions. Default.

ST_Average — Average of state transitions.

WSA_Peak — Peak of weighted switching activity.

WSA_Average — Average of weighted switching activity.

- **-INSTance *instance.pathname***

An optional switch and string that orders test patterns based on switching activity associated with a specified instance.

The -instance switch can only be used when -capture_power or -shift_power is specified.
The instance.pathname can only specify a single instance.

- **-MODule *module.name***

An optional switch and string that orders test patterns based on switching activity associated with a specified module. The module.pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

The -module switch can only be used when -capture_power or -shift_power is specified.

- **-SHIft_power {LOAD | RESPonse}**

Optional switch and literal that orders test patterns based on switching activity during shift.
Literal options include:

LOAD — based test pattern loading.

RESPonse — based on circuit response.

- **-ASCending | -DESCending**

Optional switch that orders test patterns with respect to the amount of switching activity. By default test patterns are ordered in an ascending order.

- **-REMOVE_ineffective_patterns ON | OFF**

Optional switch that enables or disables the removal of ineffective patterns which do not detect additional faults during the order pattern analysis. If you set this switch to ON, you will combine the benefits of the order_patterns and compress_patterns commands. By default the switch is set to OFF so that no patterns will be removed after the order_patterns command executes.

Examples

Example 1

Total fault coverage is 98%, and there are 2,500 patterns in memory. The following example reorders patterns in the internal pattern set until the difference between the number of patterns that is needed to achieve 93% (98 - 5) fault coverage in the current pass and the previous pass is less than 1% of the number of patterns in the previous pass:

```
order_patterns auto 5
```

Example 2

The following example creates test patterns, and then reorders them based on the average weighted switching activity in capture cycles:

```
create_patterns
order_patterns -capture_power wsa_avg ascending
// Warning: Pattern classification is turned off.
```

Related Topics

[add_processors](#)
[report_patterns](#)
[compress_patterns](#)
[set_pattern_buffer](#)
[set_pattern_classification](#)
[set_pattern_filtering](#)

printenv

Context: unspecified, all contexts

Mode: all modes

Prints out the values of the Linux environment variables.

Usage

```
printenv
```

Description

Prints out the values of the Linux environment variables.

The printenv command allows the Linux printenv command to be available as a common DFT command, for convenience in displaying Linux environment variables. Linux environment variables are automatically available as variable references within the tool.

Arguments

None

Examples

The following example prints out the values of the Linux variables in the environment:

```
printenv
```

process_dft_specification

Context: dft (with no sub context)

Mode: setup, analysis

Validates and processes the content contained in a DftSpecification wrapper.

Usage

```
process_dft_specification [id] [-no_insertion [-design_name design_name]]  
[-validate_only] [-transcript_insertion_commands]
```

Description

Validates and processes the content contained in a DftSpecification wrapper.

If you specify the [-validate_only](#) option, this command only validates the Dft specification. If you specify the [-no_insertion](#) option, the command validates and generates the Dft components. When used without either of these two options, the command validates the specification, generates the Dft components such as the RTL and the ICL description, and inserts them into the design. The insertion can be done at the netlist or the RTL level depending on the [set_context -no_rtl](#) versus [-rtl](#) option that was specified when setting the context.

If the DftSpecification being processed has a [HostScanInterface/Interface](#) consisting of only ports on the current design and the [-no_insertion](#) option is not specified, the [add_icl_scan_interfaces](#) and the [set_icl_scan_interface_ports](#) commands are issued during the insertion stage such that if you perform ICL extraction after the insertion, the ScanInterface will be named using the *id* of the HostScanInterface contained in the DftSpecification.

The data created by this command is stored inside the directory returned by the [get_tsdb_output_directory](#) command which defaults to [./tsdb_outdir](#) unless it has been changed by the [set_tsdb_output_directory](#) command.

See the “[Tesson Shell Data Base \(TSDB\)](#)” section for more information about the output directory structure.

Note

 For compatibility with the Verilog and VHDL languages, specified escaped identifiers in the DftSpecification can be terminated with either a whitespace or a backslash. This implies that backslashes are not allowed within Verilog escaped identifiers and whitespaces are not allowed within VHDL escaped identifiers even though the languages support it.

process_dft_specification.post_insertion Procedure

When performing the insertion (specifically when neither the [-validate_only](#) nor [-no_insertion](#) switch is specified), the modified design is written out inside the “[tsdb_output_directory/dft_inserted_designs](#)” directory. If you need to perform custom editing between the insertion of the Dft components and the saving of the design, you can define a Tcl proc with the name

“process_dft_specification.post_insertion” and it will automatically be called after all Dft components have been inserted and before the write_design command is invoked.

To develop this type of procedure, execute all of your custom editing commands interactively in Tessent Shell in the insertion mode. When you are finished, create the Tcl proc with those commands in it; the Tcl proc will then be called when rerunning process_dft_specification. If you need to instantiate new modules inside the design as part of this Tcl proc, you must read them in using read_verilog or read_vhdl in setup mode prior to invoking process_dft_specification.

The process_dft_specification.post_insertion procedure must have two arguments as shown in the following example. The first argument holds the configuration element object that is currently being processed. The second argument holds a list of option value pairs corresponding to the options that were used when the process_dft_specification command was called.

```
proc process_dft_specification.post_insertion { root_wrapper args } {  
    # "$root_wrapper" is the config_element object being processed.  
    # "get_config_value $root_wrapper -id <0>" returns the design_name  
    # "get_config_value $root_wrapper -id <1>" returns the id  
    # "$args" is a list of option value pairs that reflects the options used  
    # when the process_dft_specification command was called.  
    array set extra_args $args  
  
    # $extra_args(-instrument_prefix) returns the value of that option  
}
```

Licensing

Table 6-6 lists the available contexts and sub-contexts supported by each license. The -license argument of the [set_context](#) command can be used to specify a particular license to use. By default, Tessent Shell acquires the least expensive license available that provides the requested functionality. License names for MemoryBIST and BoundaryScan are accepted without the -LV or -TS suffix and will default to the LV version of the license.

The TESSENT_LICENSE_ORDER environment variable modifies the priority and order of the available licenses. Any available licenses not explicitly listed in the value of the TESSENT_LICENSE_ORDER environment variable are appended to the list in their original order.

You can refer to the [set_license_queue_timeout](#) and [get_license_queue_timeout](#) commands for more information about managing the license queuing feature.

Arguments

- *id*

An optional string that specifies the DftSpecification wrapper that is to be processed. As described in the [DftSpecification](#) wrapper syntax, the wrapper is identified by the two strings design_name and id. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(design_name,*), you can omit the id from the

process_dft_specification command. However, if several DftSpecification wrappers matching DftSpecification(design_name,*) exist in memory, you must specify the appropriate id when executing the process_dft_specification command. Refer to the following -design_name option to understand how the design_name value is obtained.

- **-no_insertion**

An optional switch that specifies to not perform the insertion of the Dft components into the design. When this option is specified, no design is needed because no insertion is performed. Some validations, such as verifying that the specified parent_instance properties in the [Sib](#), [ScanMux](#) and [Tdr](#) wrappers actually exist in the design, are only performed when this switch is omitted. If you execute the process_dft_specification command without this option and the [set_current_design](#) command has not yet been issued, an error is generated.

- **-design_name *design_name***

An optional switch and string pair that specifies the design_name value used to identify the [DftSpecification](#) wrapper when the -no_insertion option is specified. The -design_name value defaults to the current design if it has been set with the [set_current_design](#) command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

When the id is specified, if exactly one wrapper matches “DftSpecification (*,<id>)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (*,*)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- **-validate_only**

An optional switch that specifies to only perform the validation of the DftSpecification and to skip Dft element generation and insertion. Using the -no_insertion switch with the -validate_only option disables a few checks such as verifying that the specified parent_instance properties in the [Sib](#), [ScanMux](#) and [Tdr](#) wrappers actually exist in the design.

- **-transcript_insertion_commands**

An optional switch that specifies to enable the transcription of the insertion commands as they are being performed. This option can generate a large number of transcribed commands but you may find it useful to see the instantiation and connection commands created by the process_dft_specification command.

Examples

The following example reads a design and a DftSpecification file, and then processes the DftSpecification. The context is changed to the patterns -ijtag context to extract the ICL of the design and generate simulation patterns. This example uses the DftSpecification shown in the example of the [ScanMux](#) wrapper documentation. Notice the use of the iNote

“tessent_pragma verilog_insert” to inject native Verilog commands into the test bench at specific times to directly monitor the states of internal signals.

```
#----- Insert IJtag Network -----
set_context dft -no_rtl
set_tsdb_output_directory scanMux_ex1_outdir
read_verilog ./data/MyCore.v
set_current_design MyCore
read_config_data ./data/ScanMux_ex1.dft_spec
process_dft_specification

#----- Extract ICL -----
set_system_mode setup
set_context patterns -ijtag
set_system_mode analysis
write_icl -out ScanMux_ex1.icl

#----- create Simulation Patterns -----# This method is left here as a way to
perform manual test bench generation but the new
# recommended method is shown below and makes use of the create_dft_specification and
# process_patterns_specification commands.

set dut "MyCore_ScanMux_ex1_v_ctl.MyCore_inst"
open_pattern_set p1
iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b0 and end_in_pause"
iWrite tessent_rtl_tdr_orange_inst.tdr 0b0
iApply -end_in_pause
iNote "iRead tessent_rtl_tdr_orange_inst.tdr 0b0"
iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b1 and end_in_pause"
iRead tessent_rtl_tdr_orange_inst.tdr 0b0
iWrite tessent_rtl_tdr_orange_inst.tdr 0b1
iApply -end_in_pause
iNote "tessent_pragma verilog_insert \$display($time, \"ns:\"
measured tessent_rtl_tdr_orange_inst.ijtag_sel = %b\",\
\$dut.tessent_rtl_tdr_orange_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display($time, \"ns:\"
measured tessent_rtl_tdr_green_inst.ijtag_sel = %b\",\
\$dut.tessent_rtl_tdr_green_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display($time, \"ns:\"
measured tessent_rtl_tdr_red_inst.ijtag_sel = %b\",\
\$dut.tessent_rtl_tdr_red_inst.ijtag_sel);"
close_pattern_set

write_patterns ScanMux_ex1.v -ver
```

The manual method for creating simulation shown above is no longer needed nor recommended because you can now run the [create_patterns_specification](#) and [process_patterns_specification](#) commands to automate the process. Immediately after running extract_icl, you can issue the commands below and the signoff simulation patterns will be created including the [ICLNetworkVerify](#) wrapper to automatically verify the entire IJTAG network. The [run_testbench_simulations](#) and the [check_testbench_simulations](#) command automatically manage the invocation of the external simulator.

```
set spec [create_patterns_specification]
process_patterns_specification
run_testbench_simulations
check_testbench_simulations
```

Related Topics

[apply_specification_defaults](#)
[create_dft_specification](#)
[display_specification](#)
[get_tsdb_output_directory](#)
[read_config_data](#)
[set_tsdb_output_directory](#)

process_patterns_specification

Context: patterns

Mode: all modes

Validates and processes the content of the PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

Usage

```
process_patterns_specification [pattern_id] [-validate_only] [-unprocessed_only]  
[-config_objects config_object]
```

Description

Validates and processes the content of the PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

You typically need an elaborated ICL design with the Verilog interface view to run this command; the only exception is when you are using the “BSDL-Only Flow” which is described in [this section](#). The *design_name* identifier is extracted from the top ICL module. The *design_id* is extracted from the [tessent_design_id](#) attribute, described in [icl_module](#), which is stored in the top ICL module when the top ICL module is created using ICL extraction.

You use the Verilog interface view to make sure the pattern has access to all the ports of the design. If you only have the ICL view the patterns are limited to only the port defined in the ICL. For boundary Scan patterns or if you have [AdvancedOptions/ConstantPortSettings](#) to define, you must read in the top level Verilog module such that all ports of the top module are available. You can use [read_verilog](#) or [read_vhdl](#)-interface_only to load the interfaced top module. With the [Tessent Shell Data Base \(TSDB\)](#) open, a more automated and easier way to do that is to use the [read_design](#)-view interface command. It will load the interface and the ICL view directly from the opened TSDB.

If the top ICL module was not generated with ICL extraction, it may be missing the [tessent_design_id](#) attribute. In this case, the *design_id* is taken from the one specified with the [set_context](#)-*design_id* option; the *pattern_id* can be specified as the first argument to the command. When unspecified, it defaults to the *pattern_id* of the PatternsSpecification wrapper found in memory when there is only one match for PatternsSpecification(*design_name*, *design_id*, *).

If the PatternsSpecification wrapper was created manually and only includes Patterns wrappers containing only TestStep/BoundaryScan wrappers with the [bsdl_file](#) property pointing to a BSDL file, you are using what is referred to as the “BSDL only flow”. In this case, having an elaborated ICL module is not needed. If you only have one PatternsSpecification wrapper in memory, it will be used automatically. If you have many PatternsSpecification wrappers in memory, you must point to one of them using the [-config_objects](#) option.

By default and when generating patterns, any signal that is classified as unused for the pattern being written is removed from the pattern file, reducing the size of the pattern file or of the Verilog testbench. To disable this behavior, set the [ALL_EXCLUDE_UNUSED](#) keyword to 0.

The simulation of ICL network patterns supports the graybox for the sub-physical blocks listed in the [LowerPhysicalBlocksInstances](#) wrapper within the [SimulationOptions](#) wrapper.

If you specify the [-validate_only](#) option, the command only validates the Patterns specification and does not generate any patterns. Without the [-validate_only](#) option, the validation is done for each Patterns wrapper and only the error-free ones are processed.

The data created by this command is stored inside the [patterns](#) directory located in the directory returned by the [get_tsdb_output_directory](#) command which defaults to `./tsdb_outdir` unless it has been changed by the [set_tsdb_output_directory](#) command. See the “[Tessent Shell Data Base \(TSDB\)](#)” section for more information about the TSDB output directory structure. The patterns files are the Verilog test benches when the usage property inside the PatternsSpecification is “signoff”. It is the patterns in the selected format specified by the [manufacturing_patterns_format](#) property when the usage is set to [manufacturing_test](#). In either case, the `.pdl` file is always generated which shows the retargeted PDL that was used to create the patterns file.

The `process_patterns_specification` command is destructive as follows:

- All input constraints are deleted.
- The value of attribute `loadboard_pull_resistor` is reset on all ports.
- All loadboard loopback pairs are deleted.
- All added clocks are deleted.

The tool performs this destruction as follows:

- At the very start of `process_patterns_specification` so that only the settings in the PatternsSpecification affect pattern generation. For example, if your dofile defined loopback pairs with the [add_loadboard_loopback_pairs](#) command, the tool deletes these; however, the tool uses loopback pairs if you have defined such in the [Loopbacks](#) wrapper.
- At the very end of `process_patterns_specification` so that the settings in the PatternsSpecification do not affect subsequent calls to the [write_patterns](#) command.

Controlling Ports Used in Pattern and Testbench Generation

You can control the full list of ports that the tool uses to generate the patterns and simulation testbench by using the “[set_write_patterns_options -existing_used_ports](#)” command and switch prior to issuing the `process_patterns_specification` command.

The syntax is as follows:

```
set_write_patterns_options -existing_used_ports ports
```

Use this command with caution as the list of *ports* is considered an absolute collection. If the list of *ports* is missing ports that should be toggling or sampled by the pattern, the ports should be part of the *ports* collection. As a best practice, you should remove the unwanted ports instead of trying to select those ports that should appear in the pattern/testbench. The following example demonstrates how to do this in a safe manner:

```
set allExistingPorts [get_write_patterns_options -existing_used_ports]
set ports2Remove [get_ports [list sel* vdd*]]
set portsNeeded [remove_from_collection $allExistingPorts $ports2Remove]
set_write_patterns_options -existing_used_ports $portsNeeded
process_patterns_specification
```

In the example, the tool removes the ports with names beginning with “sel” and “vdd” from the default list of *ports* seen after design elaboration.

If the tool removes ports needed for the generated pattern/testbench to be effective, the result can be deceiving. For example, removing the “TDI” and “TDO” port on a chip with a TAP will end up in a pattern/testbench that will still have the annotations but no actual data will be scanned in/out, and there will be no comparison to expected values.

The *process_patterns_specification* command ignores the specified *ports* collection for any BoundaryScan test step that uses a [LoadBoardInfo](#) wrapper.

When you specify a collection of *ports*, the *process_patterns_specification* command issues the following note:

```
// Note: The following ports were selected using
// 'set_write_patterns_options -existing_used_ports <ports>':
//   { <port_name>... }.
//   Only those selected ports will appear in the generated pattern
//   files which may render them unusable if the above list is lacking
//   ports that are needed.
//   The above selection won't have an effect on boundary scan
//   patterns that use LoadBoardInfo.
```

Arguments

- *pattern_id*

Specifies the string corresponding to the *pattern_id* of the [PatternsSpecification\(design_name,design_id,pattern_id\)](#) wrapper to process.

- *-validate_only*

An optional switch that specifies to only validate the patterns specification and to not generate the patterns.

- **-unprocessed_only**

An optional switch that instructs the process_patterns_specification command to consider only those Patterns wrappers that have not been processed by a previous call to process_patterns_specification in the same Tesson Shell session. Once all the Patterns have been processed, the next call to process_patterns_specification with the -unprocessed_only switch will not process anything. It is possible to reset this behavior by doing the following:

- By reading-in a new [PatternsSpecification](#) or re-creating one.
- By resetting the attribute `is_unprocessed` on selected Patterns wrappers. Refer to the [reset_attribute_value](#) command.

- **-config_objects config_objects**

An optional switch and value pair that specifies a collection or list of config element names found inside one or many [PatternsSpecification\(design_name,design_id, pattern_id\)](#) wrappers. When the config objects are elements below a Patterns wrapper, the parent [Patterns](#) wrapper is used instead. When a config element is a PatternsSpecification or wrapper, all Patterns wrappers below it are used.

This option is used in the BSDL only flow to reference a [PatternsSpecification](#) wrapper. It can also be used to replace the -select option. You can use the full power of the [get_config_elements](#) command to find the Patterns wrapper you want to process and pass the collection to the process_patterns_specification command with this option.

Examples

The following example uses the process_patterns_specification command to validate only the patterns with the name “MemoryBist_*” in the [PatternsSpecification](#) that were automatically generated by the [create_patterns_specification](#) command and modified using the configuration data editing and introspection commands listed in [Table 10-1](#).

```
set spec [create_patterns_specification]
set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
foreach_in_col mbist_pat $mbist_pats {
    set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
    read_config_data -before $first_test_step -from_string {
        SimulationOptions {
            SimulationMacros {
                DEBUG;
            }
        }
    }
}
process_patterns_specification -validate_only -config_objects $mbist_pats
set spec [create_patterns_specification]
set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
foreach_in_col mbist_pat $mbist_pats {
    set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
    read_config_data -before $first_test_step -from_string {
        SimulationOptions {
            SimulationMacros {
                DEBUG;
            }
        }
    }
}
process_patterns_specification -validate_only -config_objects $mbist_pats
```

Related Topics

[apply_specification_defaults](#)
[create_patterns_specification](#)
[get_tsdb_output_directory](#)
[read_config_data](#)

Chapter 5

Command Dictionary (R)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual.

Table 5-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode <u>Uncollapsed</u> Collapsed	A boldface font indicates a required argument.
[]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
<i>Italic</i>	dofile <i>filename</i>	An italic font indicates a user-supplied argument.
{ }	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	Braces enclose arguments to show grouping. Do not enter the braces.
	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort <u>ON</u> <u>OFF</u>	An underlined item indicates either the default argument or the default value of an argument.
...	add_clocks <i>off_state</i> <u>pin_list</u> ...	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Command Descriptions

This section describes all commands that begin with the letter R that are available in Tesson Shell. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 5-2. Commands R

Command	Description
range_collection	Returns one or more adjacent elements from a collection and creates a new collection.
read_cell_library	Loads one or more cell libraries into the tool.
read_config_data	Reads the content of configuration data files or a string into the Tesson Shell environment.
read_core_descriptions	Reads the specified Tesson core description (TCD) files.
read_cpf	Loads a CPF (Common Power Format) file and removes any previously loaded power data.
read_def	Reads in the DEF file so that the tool can extract the coordinates of instances in the design and the dimensions of the design.
read_design	Reloads a design after DFT insertion to perform analysis or a future DFT insertion pass.
read_failures	Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.
read_fault_sites	Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.
read_faults	Updates the current fault list with the faults contained in the specified fault file or from the Tesson Shell Database (TSDB).
read_flat_model	Loads a flat model into the tool from a specified filename or from the Tesson Shell database (TSDB).

Table 5-2. Commands R (cont.)

Command	Description
<code>read_icl</code>	Reads ICL files into the internal ICL database.
<code>read_lef</code>	Reads in the LEF file so that the tool can extract the dimensions of the module.
<code>read_liberty</code>	A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules.
<code>read_modelfile</code>	Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.
<code>read_patterns</code>	Loads the external pattern set from the specified file or from the Tesson Shell database (TSDB).
<code>read_procfile</code>	Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.
<code>read_sdc</code>	Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.
<code>read_sdf</code>	Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.
<code>read_upf</code>	Loads a UPF (Unified Power Format) file and removes any previously loaded power data.
<code>read_verilog</code>	Reads one or more Verilog files into the specified or default logical library.
<code>read_vhdl</code>	Reads one or more VHDL files into the specified or default logical library.
<code>read_visualizer_preferences</code>	Reads a DFTVisualizer preferences file and sets current preferences as described in the file.
<code>read_window_contents</code>	Reads the contents of the specified file into a DFTVisualizer window.
<code>register_attribute</code>	Registers a new user-defined attribute.
<code>register_callback</code>	Register a Tcl proc which is to automatically run before or after a built-in feature of Tesson Shell.
<code>register_drc</code>	A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the analyze_drcViolation command and the Tesson DFTVisualizer GUI.
<code>register_drc_class</code>	Registers a new class of DRC rules such that a DRC of that class can be registered

Table 5-2. Commands R (cont.)

Command	Description
register_static_dft_signal_names	Registers new static DFT signal names and their attributes.
register_tcl_command	Registers a Tcl command and makes it available in the global name space.
remove_from_collection	Removes objects found in obj_spec from a collection, resulting in a new collection.
rename_instance	Renames the leaf name of an instance.
rename_net	Renames the leaf name of a net.
replace_instances	Replaces the module definition of an instance with a new module definition.
rename_module	Assigns a new name to an existing module.
report_aborted_faults	Displays information on some potentially testable faults that remain undetected (UD) after the ATPG process.
report_atpg_constraints	Displays all the current ATPG state restrictions and the pins on which they reside.
report_atpg_functions	Displays all the current ATPG function definitions.
report_atpg_timing	Reports the current ATPG timing setup.
report_attributes	Creates a human readable report for the set of attributes based on the usage options.
report_bisr_repair_register_icl_instances	Reports a list of BISR repair registers found in the design.
report_bist_capture_ranges	Displays a list of the currently defined BIST capture ranges.
report_black_boxes	Displays information on blackboxes and undefined models.
report_boundary_scan_ports	Displays a table listing all of the ports with their inferred or specified cell_options.
report_bus_data	Displays the bus data information for either an individual bus gate or for the buses of a specific type.
report_bypass_chains	Displays the defined EDT bypass scan chain connections.
report_capture_handling	Displays any special data capture handling currently in use.
report_capture_procedures	Displays the cyclized information for a specified named capture procedure.
report_cell_constraints	Displays a list of the constrained cells.

Table 5-2. Commands R (cont.)

Command	Description
<code>report_cell_models</code>	Displays a list of either all cell models or the DFT library models associated with the specified cell type.
<code>report_chain_masks</code>	Reports the list of masked scan chains and their associated unload value.
<code>report_clock_controller_pins</code>	Reports the clock controller pins specified with the <code>set_clock_controller_pins</code> command.
<code>report_clock_controls</code>	Displays current configuration information for clock control definitions.
<code>report_clock_domains</code>	Displays all clocks and their corresponding clock domain numbers.
<code>report_clock_gating</code>	Reports specified clock gating instances or modules in the design.
<code>report_clocks</code>	Displays a list of all user-defined clocks and SDC-defined clocks.
<code>report_compactor_connections</code>	Displays defined scan chain connections for each EDT scan channel output.
<code>report_config_data</code>	Reports the content of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to a file using the <code>write_config_data</code> command.
<code>report_config_messages</code>	Reports messages associated to a configuration element. When the <code>-hierarchical</code> option is used, it also reports the messages associated to elements below the specified wrapper.
<code>report_config_syntax</code>	Reports the legal configuration syntax for a specified configuration object.
<code>report_context</code>	Reports the current context as specified by the <code>set_context</code> command and any inferred subcontexts.
<code>report_control_signals</code>	Displays the rules checking results for control signals consisting of clocks added with the <code>add_clocks</code> command and pins identified for gating scannable memory elements with test logic.
<code>report_core_descriptions</code>	Reports all core description files.
<code>report_core_instance_parameters</code>	Reports the instrument configuration parameters and their values that were specified when the core instance was added with the <code>add_core_instances</code> commands <code>-parameter_values</code> switch.

Table 5-2. Commands R (cont.)

Command	Description
report_core_instances	Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.
report_core_parameters	Reports all core parameters and their values.
report_design_sources	Reports the pathnames or file extensions previously specified with the set_design_sources command.
report_dfm_rules	Returns a tabular report of the DFM rules contained in the layout database.
report_dft_clock_enables	Reports the currently defined DFT clock enables.
report_dft_clock_muxes	Reports the currently defined DFT clock muxes that were previously added using the add_dft_clock_muxes command.
report_dft_control_points	Reports the currently defined DFT control points that were previously added using the add_dft_control_points command.
report_dft_modal_connections	A command to report the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands
report_dft_signal_names	Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration.
report_dft_signals	Reports all the DFT signals added or to-be-created in the design.
report_diagnosis	Writes the scan diagnosis report to standard I/O.
report_display_instances	Shows netlist information for the specified instances displayed in the Flat Schematic, Hierarchical Schematic, or Data window.
report_drc_rules	Displays either a summary of DRC violations (fails) or violation occurrence message(s).
report_edt_abort_analysis	Reports the results of EDT Aborted fault analysis.
report_edt_blocks	Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.
report_edt_configurations	Displays configuration information for the current compression logic.
report_edt_finder	Reports information about the EDT logic contained in a design.

Table 5-2. Commands R (cont.)

Command	Description
<code>report_edt_instances</code>	Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.
<code>report_edt_lockup_cells</code>	Displays a tabular report of the lockup cells in the EDT logic.
<code>report_edt_pins</code>	Displays the currently defined names and inversion status of EDT channel and control pins.
<code>report_environment</code>	Displays the current value(s) of many of the most common “set_...” commands.
<code>report_external_simulator</code>	Displays the shell command the tool will use to invoke an external simulator for verifying patterns.
<code>report_failures</code>	Simulates the current design and test patterns and reports the failures in a pattern-based format.
<code>report_false_paths</code>	Displays false paths previously defined with the add_false_paths command.
<code>report_fault_sites</code>	Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.
<code>report_faults</code>	Displays fault information from the current fault list.
<code>report_feedback_paths</code>	Displays a report of currently identified feedback paths.
<code>report_flattener_rules</code>	Displays either a summary of all the flattening rule violations or the data for a specific violation.
<code>report_gates</code>	Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.
<code>report_graybox_statistics</code>	Reports the statistics gathered by graybox analysis.
<code>report_icl_extraction_options</code>	Provides a human readable report of the ICL extraction options.
<code>report_icl_modules</code>	Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.
<code>report_iclock</code>	Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.
<code>report_id_stamp</code>	Displays the unique identifier that the tool assigns each internal pattern set.
<code>report_iddq_exceptions</code>	Displays current exceptions to Iddq restrictions.

Table 5-2. Commands R (cont.)

Command	Description
<code>report_ijtag_instances</code>	Reports options of instances for which there is a matched ICL module.
<code>report_ijtag_logical_connections</code>	Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports.
<code>report_ijtag_retargeting_options</code>	Reports the type and value of each option of the previous <code>set_ijtag_retargeting_options</code> command or their default values if the <code>set_ijtag_retargeting_options</code> command was not issued.
<code>report_input_constraints</code>	Displays the present constraint status of primary input pins.
<code>report_insert_test_logic_options</code>	Reports the prefix/infix values that will be used when logic is created by the <code>insert_test_logic</code> command.
<code>report_iprocs</code>	Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module.
<code>report_layout_core_information</code>	In hierarchical layout-aware diagnosis, lists the instance pathnames of a core stored in an instance-aware core-level LDB.
<code>report_layout_files</code>	Returns the flat models and pattern sets associated with RCD constants stored in the LDB.
<code>report_layout_rules</code>	Provides detailed reporting of layout rule violations.
<code>report_lbist_configuration</code>	Reports the global LogicBIST controller configuration parameters.
<code>report_lbist_pins</code>	Reports the LogicBIST controller pins using the <code>set_lbist_pins</code> command.
<code>report_lfsr_connections</code>	Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.
<code>report_lfsrs</code>	Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).
<code>report_licenses</code>	Reports the license features currently checked out by the tool.
<code>report_lists</code>	Displays the list of pins whose values the tool will report during simulation.
<code>report_loadboard_loopback_pairs</code>	Reports the loopback connections and the ac_delay, if any, that were created using the <code>add_loadboard_loopback_pairs</code> command.
<code>report_loops</code>	Displays information about circuit loops.

Table 5-2. Commands R (cont.)

Command	Description
<code>report_lpct_configuration</code>	Reports the current low pin count test controller (LPCT) parameters to the standard display output.
<code>report_lpct_pins</code>	Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.
<code>report_measure_cycles</code>	Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.
<code>report_memory_cluster_configuration</code>	Reports the configuration of cluster memories tested in subsequent controller steps for a specified memory cluster and memory access level.
<code>report_memory_identification</code>	Reports the validation results of the memory module matching performed from the name patterns specified with the <code>set_memory_identification_options</code> command.
<code>report_memory_instances</code>	Reports the options of all instances for which there is a matched TCD Memory description.
<code>report_memory_repair_groups</code>	Reports information about the repair groups from a DftSpecification.
<code>report_mismatch_sources</code>	Displays the sources of simulation mismatches.
<code>report_misrs</code>	Reports the MISRs you have added to your design.
<code>report_misr_connections</code>	Reports MISR connections.
<code>report_module_matching</code>	Reports design modules and their matching modules of the specified format, if the <code>set_current_design</code> command has already been executed.
<code>report_module_matching_options</code>	Reports the current settings defined by the <code>set_module_matching_options</code> command.
<code>report_multicycle_paths</code>	Displays multicycle path definitions you previously read into the tool using the <code>read_sdc</code> command.
<code>report_multiprocessing_options</code>	Displays the values of all variables used when executing multiprocessing commands for ATPG or fault simulation.
<code>report_nofaults</code>	Displays the nofault settings for the specified pin pathnames or pin names of instances.
<code>report_nonscan_cells</code>	Displays the non-scan cells whose model type you specify.
<code>report_nonscan_models</code>	Displays the sequential non-scan model list.
<code>report_notest_points</code>	Displays all the circuit points for which you do not want the tool to insert controllability and observability.
<code>report_output_masks</code>	Displays a list of the currently masked primary output pins.

Table 5-2. Commands R (cont.)

Command	Description
report_pattern_filtering	Displays the current pattern filtering configuration.
report_pattern_sets	Creates a report of a specified pattern set or of all pattern sets.
report_patterns	Displays information about patterns.
report_power_data	Reports the power data currently loaded in the system from either a read_cpf or read_upf command.
report_power_metrics	Displays shift and capture power metrics for specified test patterns.
report_primary_inputs	Displays a list of the primary inputs of a circuit.
report_primary_outputs	Displays the specified primary outputs.
report_procedures	Displays the specified procedure.
report_processors	Displays information about the processors used for the current multiprocessing environment.
report_read_controls	Displays all of the currently defined read control lines.
report_read_verilog_options	Reports options specified with the set_read_verilog_options command.
report_register_value	Reports register information for each register name.
report_resources	Displays the machine resources used by the active tool session.
report rtl_to_gates_mapping	Reports the name-mapping rules for RTL to post-synthesis/post-layout name mapping.
report_run_synthesis_options	Reports the values for all options set with the set_run_synthesis_options command.
report_scan_cells	Displays a report on the scan cells that reside in the specified scan chains.
report_scan_chains	Displays a report on all the current scan chains.
report_scan_elements	Reports the scan elements that are created by the tool for scan insertion purposes.
report_scan_enable	Reports on scan_enable signals and associated scan chains.
report_scan_groups	Displays a report on all the current scan chain groups.
report_scan_models	Displays the sequential scan models currently in the scan model list.
report_scan_modes	Reports all available scan modes after they are inserted by Tesson Scan.

Table 5-2. Commands R (cont.)

Command	Description
<code>report_scan_polygons</code>	Traces scan chains and reports the cell and output net polygons for inter-scan cell logic located along the traces.
<code>report_scan_segments</code>	Reports the scan segment declarations that are defined either via <code>add_scan_segments</code> command or <code>tcd_scan</code> files.
<code>report_scan_volume</code>	Displays the volume of scan data used by the pattern set.
<code>report_seq_transparent_procedures</code>	Displays a list of <code>seq_transparent</code> test procedures along with the associated data that you specify.
<code>report_sequential_fault_depth</code>	Displays estimates of maximum test coverage possible with different sequential depth settings.
<code>report_shift_registers</code>	Reports the identified shift registers in the design after switching to analysis mode.
<code>report_silicon_insight_result</code>	Reports the summary diagnosis results for failing flops and pseudo failing flops for ATPG.
<code>report_simdut_faults</code>	Reports the SimDUT faults.
<code>report_simulation_contexts</code>	Lists the available simulation contexts and indicates the current simulation context.
<code>report_simulation_forces</code>	Lists the active forces on the specified <code>gate_pin</code> objects.
<code>report_simulation_library_sources</code>	Reports the values specified with the <code>set_simulation_library_sources</code> command.
<code>report_static_dft_signal_settings</code>	Reports the setting specified or implied with the <code>set_static_dft_signal_values</code> command.
<code>report_statistics</code>	Displays a detailed statistics report to the screen.
<code>report_synchronous_clock_groups</code>	Displays a list of all user-defined synchronous clock groups.
<code>report_tcd_ldb_validation</code>	Verifies that the core instance pathnames and hierarchy in the TCD matches the core instance pathnames and hierarchy in the LDB.
<code>report_tcl_shell_options</code>	Lists the currently specified Tcl shell options.
<code>report_test_end_icall</code>	Reports a single view of all existing iCalls specified with the <code>set_test_end_icall</code> command.
<code>report_test_logic</code>	Displays the test logic that the tool added during the scan insertion process.
<code>report_test_points</code>	Displays test points that are inserted with the <code>insert_test_logic</code> command.

Table 5-2. Commands R (cont.)

Command	Description
report_test_point_statistics	Use this command to report statistics regarding the location of test points inside the design.
report_test_setup_icall	Reports a single view of all existing iCalls specified with the set_test_setup_icall command.
report_test_stimulus	Displays the stimulus necessary to satisfy the specified set, write, or read conditions.
report_testbench_simulation_options	Reports the default values for many options used in the run_testbench_simulations command.
report_tied_signals	Displays a list of the tied floating signals and pins.
report_timeplate	Displays the specified timeplate.
report_udfm_statistics	Reports detailed fault coverage based on the UDFM defect models.
report_version_data	Displays the current software version information.
report_wrapper_cells	Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.
report_write_controls	Displays the currently defined write control lines and their off-states.
report_write_patterns_options	Displays the current set of ports or current set of vector callbacks to use when writing patterns.
report_xbounding	Reports the X-sources and optionally the scan cells used in bounding. You must issue this command to report X-bounding.
reset_attribute_value	Resets an attribute to its default value for the design objects specified in <i>obj_spec</i> .
reset_au_faults	Reclassifies the faults in certain untestable categories.
reset_bypass_chains	Removes user-specified EDT bypass chain connections.
reset_compactor_connections	Removes user-specified EDT compactor connections.
reset_design	Clears all design setup data but keeps the design and library in memory.
reset_di_faults	Reclassifies DI faults to the UC category.
reset_open_pattern_set	Clears the content of the currently open pattern set.
reset_state	Resets the circuit status.
reset_static_dft_signal_values	Resets the static DFT signal values that were previously set using the set_static_dft_signal_values command.

Table 5-2. Commands R (cont.)

Command	Description
restore_design	Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.
run_synthesis	Invokes a synthesis manager to perform synthesis of the test logic RTL.
run_testbench_simulations	Invokes a simulation manager to run a set of simulation test benches.

range_collection

Context: unspecified, all contexts

Mode: all modes

Returns one or more adjacent elements from a collection and creates a new collection.

Usage

`range_collection collection first last`

Description

Returns one or more adjacent elements from a collection and creates a new collection.

The returned collection consists of elements first through last, inclusive. If first is greater than last, this command returns an empty collection.

The base collection remains unchanged.

Arguments

- ***collection***
A required string that specifies the collection to be searched.
- ***first***
A required integer that specifies the first index into the collection. Allowed values are from 0 to `sizeof_collection - 1`.
- ***last***
A required integer that specifies the ending index into the collection. Allowed values are from 0 to `sizeof_collection - 1`. You can also specify the value of the last argument relative to the end of the collection by using “end-<integer>”.

Examples

The following example returns a collection with the last two elements from \$mycoll.

`range_collection $mycoll end-1 end`

Related Topics

- [add_to_collection](#)
- [append_to_collection](#)
- [compare_collections](#)
- [copy_collection](#)
- [filter_collection](#)
- [foreach_in_collection](#)

[is_collection](#)
[remove_from_collection](#)
[sizeof_collection](#)
[sort_collection](#)

read_cell_library

Context: unspecified, all contexts

Mode: setup

Loads one or more cell libraries into the tool.

Usage

```
read_cell_library lib_path ... [-force]
```

Description

Loads one or more cell libraries into the tool.

This command provides functionality equivalent to the -Library tool invocation switch.

You can issue this command multiple times. In case of parsing errors, the tool clears all the library data and returns to Setup mode. After the erroneous file has been corrected and reloaded, the tool reparses all files that were successfully parsed previously.

Arguments

- ***lib_path***

A required and repeatable string that specifies the names of the files containing the Tesson Cell library descriptions for cell models in your design. You can use wildcards to specify multiple file names.

- **-force**

An optional switch to suppress the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a library using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), executing the read_cell_library command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you execute the command with the -force switch to indicate that it is acceptable to delete the setup data.

Examples

The following example loads two cell libraries:

```
read_cell_library lib1 lib2
```

Related Topics

[delete_cell_library](#)

[set_cell_library_options](#)

[write_cell_library](#)

read_config_data

Context: unspecified, dft, patterns

Mode: all

Reads the content of configuration data files or a string into the Tesson Shell environment.

Usage

```
read_config_data {filename_list | -from_string string}
    [-in_wrapper parent_wrapper]
    [-before before | -after after | -first | -last]
    [-replace]
```

Description

Reads the contents of configuration data files or a string into the Tesson Shell environment.

The configuration data file format is currently used for two released features but it will be used to describe many more specifications in the future.

The first usage is to load a dft specification and a defaults specification containing the syntax described in the [DftSpecification](#), and [DefaultsSpecification/DftSpecification](#) sections and which are used by the [get_defaults_value](#) and [process_dft_specification](#) commands.

The second usage is to load the test setup specifications described in “[Prepare the Design Under Test](#)” in the *Tesson SiliconInsight User’s Manual for Tesson Shell* to allow Tesson SiliconInsight to communicate via Desktop, SimDUT, or offline ATE modes with the design under test (DUT). In this application, Tesson Shell uses the settings in the Tesson SiliconInsight setup specification to control bring-up and usage of the Tesson SiliconInsight session.

Arguments

- *filename_list*
An optional string that specifies the name of a file or a tcl list of file names. The names can include wildcards (the * character) to allow matching multiple file names with a single pattern.
- *-from_string string*
An optional value pair that specifies a string to parse. the string can be contained in a tcl variable or typed between curly bracket.
- *-in_wrapper parent_wrapper*
An option and value pair that specifies that the read in configuration elements are to be loaded inside a specific wrapper. The *parent_wrapper* can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by or “-object”. When the *-in_wrapper* is not specified, the destination

parent wrapper is assumed to be the root of the partition unless the -before or -after option is specified.

If the specified root wrapper is a wrapper of type CSV, you can load a CSV file generated out of Excel™ where the line terminator is the carriage return instead of a semicolon.

- **-before before**

An optional value pair that specifies a configuration element before which to the new configuration elements are to be loaded. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

When before is a name, it must a name relative to the wrapper specified with the -in_wrapper option. It is a name relative to the root of the partition when the -in_wrapper option is not specified.

The -before option is mutually exclusive to the -after, -first and -last options. When none of those four option is specified, -last is assumed.

- **-after after**

An optional value pair that specifies a configuration element after which the new configuration elements are to be loaded. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

When after is a name, it must a name relative to the wrapper specified with the -in_wrapper option. It is a name relative to the root of the partition when the -in_wrapper option is not specified.

The -after option is mutually exclusive to the -before, -first and -last options. When none of those four option is specified, -last is assumed.

- **-first**

An optional that specifies the new configuration elements are to be loaded before any other element in the parent wrapper.

The -first option is mutually exclusive to the -before, -after and -last options. When none of those four option is specified, -last is assumed.

- **-last**

An optional switch that specifies the new configuration elements are to be loaded after all other elements in the parent wrapper.

The -last option is mutually exclusive to the -before, -after and -first options. When none of those four option is specified, -last is assumed.

- **-replace**

An optional switch that suppresses the error that is normally generated when loading configuration elements having the same name as an existing element in the destination

wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

Example 1

The following example loads the content of the *mydesign.dft_spec_rtl* file into the Tessent Shell environment.

```
read_config_data mydesign.dft_spec_rtl
```

Example 2

The following example reads a CSV file into a wrapper of type CSV.

```
add_config_element csv(1)
read_config_data data.csv -in_wrapper csv(1)
report_config_data csv(1)
```

Related Topics

[Configuration-Based Specification](#)

[display_specification](#)

[process_dft_specification](#)

read_core_descriptions

Context: all contexts

Mode: setup

Reads the specified Tessent core description (TCD) files.

Usage

read_core_descriptions *file_names*

Description

Reads the specified Tessent Core Description (TCD) files.

The Tessent Core Description file is used to map scan information from one level to the next for running ATPG at the higher level, and to retarget patterns for instances of the associated core. Additionally, the TCD is used to describe EDT IP instances for ATPG.

The Tessent Core Description format is also used to document the behavior of Memory modules, the embedded BoundaryScan content of a module, or the behavior of a FuseBoxInterface module. See “[Tessent Core Description](#)” on page 3755 for descriptions.

To view the contents of a Core wrapper loaded with this command, use the “[report_config_data](#) Core(module_name) -partition tcd -level 1” command to see the first level of wrappers. Then, use the “[report_config_data](#) Core(module_name)/XXX -partition tcd” command to see the content of wrapper XXX.

The TCD format contains public wrappers such as Memory, BoundaryScan, and FuseBoxInterface, and private wrappers such as AtpgMode(mode_name) and LogicBistMode(mode_name). The public wrapper formats are documented in the [Tessent Core Description](#) chapter.

Arguments

- *file_names*

A required, repeatable string that specifies the core description files to be read. The names of the files can contain wildcards. You can define the names of files to read into a Tcl list, and use the name of the Tcl list as the *file_names* argument.

Examples

The “[Retargeting Example](#)” in the *Tessent Scan and ATPG User’s Manual* demonstrates the use of this command in a design context.

Related Topics

[add_core_instances](#)

[delete_core_descriptions](#)

[delete_core_instances](#)
[report_core_descriptions](#)
[report_core_instances](#)
[write_core_description](#)

read_cpf

Context: all contexts

Mode: setup

Loads a CPF (Common Power Format) file and removes any previously loaded power data.

Usage

```
read_cpf filename [{> | >>} file_pathname]
```

Description

Loads a CPF (Common Power Format) file and removes any previously loaded power data. You can use multiple CPF files by using the native CPF “include” command.

The tools support CPF version 1.0 and 1.1.

Effects of Reading Power Data

Loading power data from a CPF or UPF file has the following effects:

- Simulation of test procedures and scan patterns take into account whether a gate is powered on during simulation.
- The tool runs the Power-Aware Rules (V Rules) when transitioning out of Setup mode.
- Scan insertion restricts scan stitching such that each scan chain includes only scan cells from one power domain.
- This command loads power data into the tool and the [report_gates](#) command enables power data reporting to show the power on (PON) or power off (POFF) as well as the power domain of the gate.
- Test point sharing is restricted based on power information.
- The power attributes provide access to the power information. The power attributes are:

Data Model	Power Attribute
Instance, Module	is_power_always_on_cell is_power_isolation_cell is_power_level_shifter_cell is_power_retention_cell
Instance, Module, Net, Pin, and Port	power_domain_name

Note

As described in the create_dft_specification command description, the DftSpecification is affected by the power domain information to make sure that groups of DesignInstances associated to different power domains are isolated by a SIB. This is done to avoid daisy chaining them with elements from a different power domain and running into issues when accessing the elements in a given power domain while the other power domains are off.

Arguments

- *filename*
A required string that specifies the name of the CPF file.
- *>file.pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.
- *>>file.pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

Example 1

This example loads a CPF file:

```
read_cpf cpf_file
```

Example 2

In this example, read_cpf loads power data into the tool and the report_gates command reports the power on (PON) status as well as the power domain of the gate.

```
read_cpf cpf_file /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff
report_gates /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff
//  /lp_case_si_rst_sms/ati_rst_sync/sync_r  sync3msfqxss1ul
//    SDI      I(PON)   /vl_sms_lp_case_si_sms_proc_sms_1_stp/
U_lp_case_si_sms_proc_bist/uu5/Z
//      D      I(PON)   /lp_case_si_rst_sms/ati_rst_sync/sync_buf/Z
//      SEN     I(PON)   /se
//      CLK     I(PON)   /lp_case_si_rst_sms/ati_rst_sync/uu1/Z
//      Q      O(PON)   /lp_case_si_rst_sms/ati_rst_sync/uu2/A  /
lp_case_si_rst_sms/and_r/B
//      in power domain PD_P2
```

Related Topics

[add_faults](#)

[delete_faults](#)

[read_faults](#)

[read_upf](#)

[report_faults](#)

[report_power_data](#)

read_def

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the `set_current_design` command.

Reads in the DEF file so that the tool can extract the coordinates of instances in the design and the dimensions of the design.

Usage

```
read_def file_name [-instance_of_current_design instance_name] [-sequential_cells ON | OFF] [-default_udm default_udm]
```

Description

Reads in the DEF file so that the tool can extract the coordinates of sequential cells, ports of the current design, instances reported by `get_memory_instances`, and lower-level physical blocks. A lower-level physical blocks is a module of type design which is placed or fixed in the DEF file.

When using this command, the physical coordinates of the memories and sequential elements are used for grouping memories during MemoryBIST insertion and for placement of Boundary Scan cells. Note that Tessent Scan does not use the DEF file for scan insertion or stitching.

The tool stores the (x,y) coordinates of top-level ports and all instances in the following attributes:

- `def_x_coordinate`
- `def_y_coordinate`

All coordinates are floating point values in microns (um).

The tools also stores the orientation of the instances in the following attribute:

- `def_orientation`

Refer to “[Instance](#)” on page 3069.

In addition, the `read_def` command also extracts the lower left and upper right coordinates of the die. The lower left coordinates (`xl`, `yl`) and the upper right coordinates (`xh`, `yh`) of the die are stored in the following attributes for the top module:

- `def_xl_coordinate`
- `def_yl_coordinate`
- `def_xh_coordinate`
- `def_yh_coordinate`

The `read_def` command also extracts lists of the x coordinates and corresponding y coordinates for the top module of the current design. This information provides the physical shape, size, and location of the die. The tool stores the lists in these attributes:

- `def_x_coordinate_list`

This list contains the x coordinates for the top module of the current design.

- `def_y_coordinate_list`

This list contains the y coordinates for the top module of the current design.

For example:

```
blockA def_x_coordinate_list=='{0 0 8040 8040 13080 13080 10120 10120}'
def_y_coordinate_list=='{0 2347400 2347400 2352640 2363120 2363120 0}'
```

An x coordinate and corresponding y coordinate form a vertex of a polygon. The polygon describes the physical shape, size, and location of the top module of the current design.

Refer to “[Module](#)” on page 3058.

Arguments

- `file_name`

A required string that specifies the name of the DEF file.

- `-instance_of_current_design instance_name`

An optional switch and value pair that is used when the current design in Tessent Shell is a sub-module of the design that existed in the synthesis or layout tool when creating the DEF file. This situation occurs when you want to perform DFT insertion on a sub-block module and the sub-block module was synthesized as part of its parent physical block, so therefore it is not possible to obtain a DEF file for the sub-block alone. In this case, the DEF file created for the parent physical block is used and you specify, with the `-instance_of_current_design` switch, the instance inside the physical block DEF file that matches the sub-block module that is set as the current design in Tessent Shell. The coordinates of the instances found below that instance in the DEF file will be attached to the corresponding instances found below the current design inside Tessent Shell. This will allow the tool to create clusters of memories and do all other physically aware placement supported by Tessent Shell. For more information about the differences between a sub-block and physical block, refer to the [set_design_level](#) command description.

- `-sequential_cells ON | OFF`

An optional switch and literal pair that specifies whether to extract the coordinates for sequential cells. The default is ON.

- `-default_udm default_udm`

An optional switch and integer pair that overrides the default dbu-per-micron value of 100 from the DEF file.

Related Topics

[read_lef](#)

read_design

Context: dft, patterns

Mode: setup

Reloads a design after DFT insertion to perform analysis or a future DFT insertion pass.

Usage

```
read_design [[design_name] [-design_identifier design_identifier] [-view view]  
[-no_hdl] | -from_dictionary from_dictionary] [-verbose] [-force]  
[-skip_existence_checks]
```

Description

The `read_design` command is used to reload a design to perform analysis or a new DFT insertion pass.

You need to use the `read_verilog` and `read_vhdl` commands to initially read your design into the tool. Once you have performed a DFT insertion pass on the design, the list of `read_verilog` and `read_vhdl` commands required to reload the modified design into the tool is stored in the `design_name.design_source_dictionary` file located in the `dft_inserted_designs` directory—see “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763.

For all subsequent reloads of the design, you use the `read_design` command to automate the reading of the current view of the design. Additionally, the ICL view is automatically loaded in all contexts. If you call the `extract_icl` command, the top level ICL module is automatically deleted to allow re-extracting it. It used to be that when you used `read_design` in the patterns `-ijtag`, the ICL view was not read in. If you then called “`set_system_mode analysis`”, the top ICL module was extracted. This no longer happens because the ICL is not extracted by the system mode transition when the ICL for the top module already exists in memory. If you use the `extract_icl` command as is recommended, the backward compatibility difference is not seen because `extract_icl` deletes the top level ICL module when it exists to allow re-extracting it.

If the top design is read in using `read_design`, then the `design_level` is also imported from the `.tcd` file found in the `dft_inserted_designs` directory.

[Figure 5-1](#) illustrates an important aspect of design views in a bottom-up flow. The figure shows a parent block instantiating two child blocks. In a bottom-up flow, you start with the golden RTL views of the parent block and the two child blocks. Once you have performed the first DFT insertion pass on the two child blocks, you may want to move up to the parent block and perform the first DFT insertion pass on the parent, and verify the DFT logic within the child blocks from the top level. The first insertion pass is typically memoryBIST and IJTAG. In [Figure 5-1](#), the blue child blocks within the blue parent block (top-right corner of the figure) illustrates the RTL view where memoryBIST and IJTAG were inserted in all three blocks. You can use this view to fully verify the memoryBIST in the parent block and within the child blocks as accessed from the IJTAG scan interface at the boundary of the parent block.

When you want to perform the second DFT insertion pass, which is typically for the logic test insertion pass, you will start again with the two child blocks. When you want to do the logic test DFT insertion pass on the parent block, you will start with the design view shown in the bottom-right corner of [Figure 5-1](#). The `read_design` command is used to achieve this task. You load the blue view of the parent block using the following command syntax:

```
read_design parent_module_name -design_id blue
```

When issued, the above command only loads the modules below the parent block but above the two child blocks. This is true for the HDL part (Verilog and/or VHDL), and it is also true for ICL. The ICL file found inside the blue TSDB `dft_inserted_designs` directory contains the complete hierarchical view of the ICL including the two child blocks. However, there is a pragma in the ICL file that separates the modules belonging to the top module and those belonging to the child blocks. The pragma is this comment is as follows:

```
//tessent pragma child_block_beginning
```

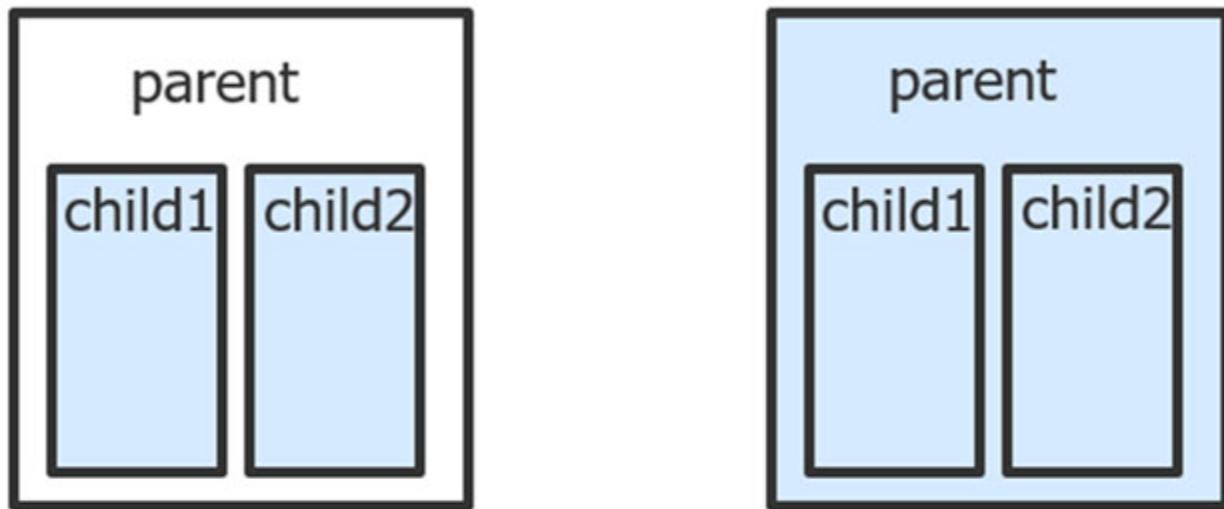
When you issue the `read_design` command with either the `-no_hdl` or the `-view full` options, it will invoke the `read_icl` command with the switch `-skip_child_blocks` such that only the ICL module found between the top level and the first layer of child blocks is loaded.

[Figure 5-1](#) illustrates the reasoning as to why this is done. When doing the second DFT insertion pass of the parent, it is the orange view of the child block that must be loaded in memory, not the blue view. If the `read_design` command loads the full ICL view of the parent module all the way down to the bottom, then the tool would load the blue ICL view of the child blocks.

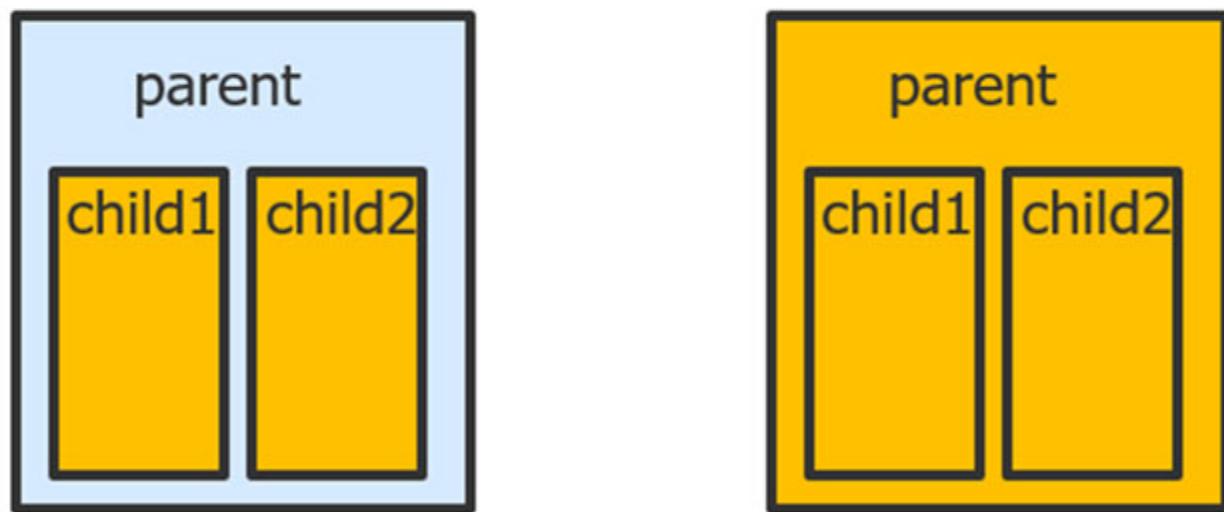
If you do not read in the child blocks before you issue the `set_current_design` command, the child block orange interface view for the HDL and their full ICL view will automatically be loaded from their TSDB `dft_inserted_designs` directory (as long as you have opened their TSDBs). You can force the loading of specific views of the child blocks by using the `read_design` command and specifying the specific `-design_id` and `-view` you want.

Figure 5-1. Design Views in a Bottom-up Flow

Insertion pass 1 (design_id = blue)



Insertion pass 2 (design_id = orange)



Arguments

- *design_name*

An optional string specifying the name of the design to reload. This value is mandatory when the design is not already elaborated and the [get_current_design](#) command returns an empty collection. When the design is already elaborated, you can omit the name of the

design as it is assumed you are trying to reload the current design. You must, however, specify the -force switch when the design is already elaborated to allow the tool to destroy the current elaborated view and reload the design.

- **-design_identifier *design_identifier***

An optional switch-value pair used to specify the *design_id* of the design you want to read in. You only need to specify this option when the design is not already elaborated and the **Tessent Shell Data Base (TSDB)** contains multiple **dft_inserted_designs** views for the *design_name* you want to read in.

A *design_id* must only use a combination of the following characters:

- Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
- Numeric. For example: “123”.
- Underscore. For example: “A_1”.

- **-view full | graybox | interface**

An optional switch-value pair used to specify which view of the design to read-in. When you specify interface, the command reads in the interface view from the TSDB which only includes the ports and the parameters definitions of the module. When you specify graybox, it loads the graybox view which was created by the **analyze_graybox** command and stored in the TSDB using the **write_design -tsdb -graybox** command. When you specify full, the command reads in the complete RTL or netlist view of the design. The command reads in the RTL view when in the context -rtl mode and the netlist view when in the context -no_rtl mode. When the -view switch is not specified, the full view is loaded. Using **read_design -view interface** is the recommended method to read-in the ICL and the interface view to use with the **process_patterns_specification** command.

- **-no_hdl**

An optional Boolean switch used to specify that you want to load the ICL and TCD views of the design but skip loading the Verilog or VHDL view. You use this switch when you got a netlist from synthesis or layout and you want to load the rest of the information from its pre-synthesis or pre-layout view as shown in [Example 3](#).

- **-from_dictionary *from_dictionary***

An optional switch-value pair used to supply a Tcl dictionary as input to the **read_design** command. When using the -from_dictionary switch, you must specify a DICTIONARY for *from_dictionary*, not a file that contains a dictionary, otherwise the tool issues an error similar to the following:

```
// Error: The supplied dictionary is not a list with an even number
// of elements.
```

The format of the Tcl dictionary is documented in the **dft_inserted_designs** section of the TSDB chapter.

Note



You cannot read in the dictionary stored in the TSDB using the -from_dictionary option.

This switch is useful to read in the design for the first time into the tool. You can have your RTL flow generate the dictionary and use `read_design -from_dictionary` to load it into the tool using the file dictionary you imported into the tool. You can also create a Tcl Dictionary by reading the design into the tool using `read_verilog` and `read_vhdl` and then calling `get_design_sources -file_dictionary` after having called the `set_current_design` command.

See “[Example 4](#)” on page 1219.

- `-verbose`

An optional Boolean switch used to request that the command transcript all the read commands used to load the design.

- `-force`

An optional Boolean switch used to disable the error you would receive if calling the `read_design` command when a design is already elaborated inside the tool.

If you call `read_design -force` and the design that was already elaborated has clocks, input constraints, and/or black boxes definitions, the tool will restore these after the newly loaded design is re-elaborated using the `set_current_design` command.

- `-skip_existence_checks`

An optional Boolean switch used to suppress the errors that the tool normally issues if the design-source dictionary contains nonexistent file or directory references. Refer to the [example](#) in the `write_design_source_dictionary` command description to understand when you should use this switch.

Examples

Example 1

This example shows how to use the `read_design` command to reload a design into the tool as it was saved after a DFT insertion pass that used a *design_id* equal to “rtl”:

```
set_context dft -rtl -design_id rtl2
read_design mytop -design_id rtl
set_current_design mytop
```

If you exited Tessent Shell after performing DFT insertion, you can use the same steps listed above to reload the design from a TSDB directly without having to perform DFT insertion again. If the TSDB resides in the same directory where Tessent Shell is executed and uses the default name (*tsdb_outdir*) then the TSDB is automatically opened. However if the TSDB has a different name or resides in a different directory then one must use the “`open_tsdb TSDB_path`” command prior to issuing the commands shown above.

Example 2

This example shows how to load the concatenated netlist after some instruments were inserted into the netlist and the instruments were synthesized using the [run_synthesis](#) command. The [process_dft_specification](#) command creates the *design_name.vg_no_instruments* view in the [dft_inserted_designs](#) directory. The [run_synthesis](#) commands synthesizes the instruments and concatenates the netlist of the synthesized instruments with the *.vg_no_instruments* netlists into a single *.vg* netlist. The “[read_design -force](#)” command reloads the *.vg* netlist into the tool.

```
process_dft_specification
set_run_synthesis_options dc_shell -startup ..synopsys_dc.setup
run_synthesis -wait
read_design -force
set_current_design
```

Example 3

The following example uses the *-no_hdl* option of the [read_design](#) command to load the information about *my_design* from the pre-synthesis TSDB directory and combine it with the post-synthesis netlist to create a complete gate level TSDB directory. This will later allow using “[read_design my_design -design gate](#)” to create patterns.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Refer to [Example 2 in run_testbench_simulations](#) for a usage example about how to use the [write_design -tsdb -softlink_netlist](#) command to create a TSDB [dft_inserted_designs](#) container for a netlist generated from synthesis or layout, and on which you want to use the [run_testbench_simulations](#) command to simulate the patterns described in your Patterns Specification.

Example 4

The following example demonstrates reading a Tcl dictionary from a saved dictionary file:

```
INSERTION> write_design_source_dictionary Chip.design_source_dictionary -replace
INSERTION> set_system_mode setup
.....
#Reread the design when you enter the setup mode.
SETUP> delete_design
SETUP> source Chip.design_source_dictionary
SETUP> read_design -from_dictionary $design_source_dictionary
```

Related Topics

[run_synthesis](#)
[get_design_sources](#)

read_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.

Usage

read_failures *failure_file* [-Pattern | -Cycle] [{> | >>} *output_file*]

read_failures *failure_file* [-Append]

Description

Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.

By default, this command displays the first 20 pattern mismatches. Use the [set_design_sources](#) command to change the number of pattern mismatches that display.

You can also use the read_failures command to display additional information about failed cells within the failure file, or use it to convert a failure file to either pattern-based or cycle-based format. This command supports both, compressed and uncompressed test patterns.

Test patterns must be WGL or STIL format for cycle-based failure files.

When read_failures -append is used, then the order in which the failure files are supplied must match the order in which the multiple patterns sets have been loaded. In the event that one of the failure files is inconsistent with the corresponding test set, an error will be issued and that failure file will not be appended to the previously loaded failure files.

Note

 If you omit the -append switch, then the tool overwrites the in-memory failure file.

Tessent Diagnosis performs a basic syntax check during parsing.

Arguments

- *failure_file*

A required string that specifies the name of the failure file to read. For failure mapping, this file is a top-level failure file generated as a result of applying top-level patterns created by scan pattern retargeting.

- -Pattern

An optional switch that displays/writes the failure file in the pattern-based format. Use this switch to convert a cycle-based failure file to a pattern based format or use it to add location information to failed cells within a pattern-based failure file.

- **-Cycle**

An optional switch that displays/writes the failure file in the cycle-based format. Use this switch to convert a pattern-based failure file to a cycle-based format or use it to add location information to failed cells within a cycle-based failure file.

- **-Append**

An optional switch that appends the *failure_file* to a previously-specified failure file and saves the appended *failure_file* in memory.

Before appending a failure file, you must issue the `read_failures` command without the `-append` switch and, subsequently, issue the `read_failures` command with the `-append` switch.

- **> *output_file***

An optional redirection operator and pathname pair used at the end of the argument list to create *or* replace the contents of *output_file*.

- **>> *output_file***

An optional redirection operator and pathname pair used at the end of the argument list to append output to the contents of *output_file*.

Examples

Example 1

The following example loads a test pattern source (*1pat.wgl*), reads in a pattern-based failure file (*padding_failures.pat*) and writes out a cycle-based failure file (*pat_cycle.fail*).

```
read_patterns 1pat.wgl  
read_failures fail_files/padding_failures.pat -cycle > results/pat_cycle.fail
```

Example 2

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_failures failure_file1  
read_failures failure_file2 -append
```

Example 3

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_patterns pat1  
read_patterns pat2 -append  
read_failures failure_file1  
read_failures failure_file2 -append
```

Example 4

The following example shows the *padding_failures.pat* file.

```

chain_test
0 chain5    0   H   L
0 chain5    1   H   L
0 chain5    4   H   L
0 chain5    5   H   L
0 chain5    8   H   L
0 chain5    9   H   L
0 chain5   12   H   L
0 chain5   13   H   L
0 chain5   16   H   L
0 chain5   88   H   L
0 chain5  100   H   L
last_pattern_tested  0

scan_test
0  chain5    2   H   L
0  chain5    4   H   L
0  chain5    6   H   L
0  chain5    8   H   L
0  chain5    9   H   L
0  chain5   11   H   L
0  chain5   12   H   L
0  chain5   13   H   L
0  chain5   28   H   L
0  chain5  100   H   L
last_pattern_tested  2

```

Example 5

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```

read_failures failure_file1
read_failures failure_file2 -append

```

Example 6

The following example shows the cycle-based failure file (*pat_cycle.fail*). The cycle-based data is written in columns from left to right: cycle id, pin name, expected value, simulated value, pattern id, chain/PO, cell number, cell path name.

```
format cycle
failures_begin
// chain test
105 dhiraj_so5 H L // pat 0 chain chain5 cell 0 uUART/reg_MODE0_CLK
106 dhiraj_so5 H L // pat 0 chain chain5 cell 1 uCNTR/reg_PORT_3_SAMPLE_P1_2_
109 dhiraj_so5 H L // pat 0 chain chain5 cell 4 uCNTR/reg_PORT_3_SAMPLE_P1_5_
110 dhiraj_so5 H L // pat 0 chain chain5 cell 5 uINTR/ix546
113 dhiraj_so5 H L // pat 0 chain chain5 cell 8 uINTR/ix476
114 dhiraj_so5 H L // pat 0 chain chain5 cell 9 uINTR/ix446
117 dhiraj_so5 H L // pat 0 chain chain5 cell 12 uINTR/ix356
129 dhiraj_so5 H L // pat 0 chain chain5 cell 24 uSTM/ix56/SFFR
...
// scan test
314 dhiraj_so5 H L // pat 0 chain chain5 cell 2 uCNTR/reg_PORT_3_SAMPLE_P1_3_
316 dhiraj_so5 H L // pat 0 chain chain5 cell 4 uCNTR/reg_PORT_3_SAMPLE_P1_5_
318 dhiraj_so5 H L // pat 0 chain chain5 cell 6 uINTR/ix106
320 dhiraj_so5 H L // pat 0 chain chain5 cell 8 uINTR/ix476
321 dhiraj_so5 H L // pat 0 chain chain5 cell 9 uINTR/ix446
323 dhiraj_so5 H L // pat 0 chain chain5 cell 11 uINTR/ix386
621 dhiraj_so5 H L // pat 2 chain chain5 cell 101 padding_cycle
failure_buffer_limit_reached all
failures_end
```

Example 7

The following example loads a test pattern source (*1pat.wgl*), reads in a cycle-based failure file (*pat_cycle.fail*) and writes out a pattern-based failure file (*cyc_pat.fail*).

```
read_patterns 1pat.wgl
read_failures fail_files/pat_cycle.fail -pattern > results/cyc_pat.fail
```

Example 8

The following example shows the pattern-based failure file (*cyc_pat.fail*). The pattern-based failure file data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name:

```
chain_test
0 chain5 0 H L // uUART/reg_MODE0_CLK
0 chain5 1 H L // uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5 4 H L // uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5 5 H L // uINTR/ix546
0 chain5 8 H L // uINTR/ix476
0 chain5 9 H L // uINTR/ix446
0 chain5 12 H L // uINTR/ix356
0 chain5 13 H L // uINTR/ix326
0 chain5 16 H L // uINTR/ix236
0 chain5 88 H L // padding_cycle
0 chain5 100 H L // padding_cycle
last_pattern_tested 0

scan_test
0 chain5 2 H L // uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5 4 H L // uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5 6 H L // uINTR/ix106
0 chain5 8 H L // uINTR/ix476
0 chain5 9 H L // uINTR/ix446
0 chain5 11 H L // uINTR/ix386
0 chain5 12 H L // uINTR/ix356
0 chain5 13 H L // uINTR/ix326
0 chain5 28 H L // uSTM/reg_WAIT_SYNC/SFFR
0 chain5 100 H L // padding_cycle
last_pattern_tested 2
```

Example 9

The following example loads a test pattern source(*1pat.wgl*), reads in a compressed pattern-based failure file (*fail_file.pat*) and writes out a compressed pattern-based failure file (*pat_2_pat*) that includes cell path names for internal scan chains corresponding to each failing EDT channel.

```
read_patterns 1pat.wgl
read_failures fail_file.pat -pat > results/pat_2_pat
```

Example 10

The following example shows the compressed pattern-based failure file (*fail_file.pat*) loaded into the tool.

```
chain test
  0  edt_channel11  2  H  L
  0  edt_channel11  3  L  H
  0  edt_channel12  0  L  H
  0  edt_channel12  4  H  L
  1  edt_channel11  0  H  L
last_pattern_tested 1
scan test
  0  /x_out[4]  H  L
  0  /x_out[2]  H  L
  0  /y_out[5]  L  H
  0  /y_out[4]  L  H
  0  edt_channel11  0  L  H
  0  edt_channel11  1  H  L
  0  edt_channel11  2  L  H
  1  /x_out[3]  H  L
  1  /x_out[2]  H  L
  1  /x_out[1]  H  L
|last_pattern_tested 1
```

Example 11

The following example shows the resulting compressed pattern-based failure file (*pat_2_pat*) output by the tool. Data is written in columns from left to right: pattern_id, channel/PO_name, channel_cycle_id, expected_value, simulated_value, chain---cell_id---cell_path_name. For compressed patterns, the location of every possible failing cell in the internal scan chains corresponding to each failing EDT channel is listed.

read_failures

```

chain_test
0 edt_channel1 2 H L // chain1---2---circle_i/reg_x_2_/DF1
                        // chain2---2---circle_i/reg_f_6_
                        // chain3---2---circle_i/reg_f_1_
0 edt_channel1 3 L H // chain1---3---circle_i/reg_x_3_/DF1
                        // chain2---3---circle_i/reg_f_7_
                        // chain3---3---circle_i/reg_f_2_
0 edt_channel2 0 L H // chain4---0---circle_i/reg_y_2_
                        // chain5---0---circle_i/reg_x_5_/DF1
0 edt_channel2 4 H L // chain4---4---circle_i/reg_y_6_
                        // chain5---4---circle_i/reg_y_1_
1 edt_channel1 0 H L // chain1---0---circle_i/reg_x_0_/DF1
                        // chain2---0---circle_i/reg_f_4_
                        // chain3---0---circle_i/reg_y_7_
last_pattern_tested 1
scan_test
0 /x_out[4]          H L // PO
0 /x_out[2]          H L // PO
0 /y_out[5]          L H // PO
0 /y_out[4]          L H // PO
0 edt_channel1 0 L H // chain1---0---circle_i/reg_x_0_/DF1
                        // chain2---0---circle_i/reg_f_4_
                        // chain3---0---circle_i/reg_y_7_
0 edt_channel1 1 H L // chain1---1---circle_i/reg_x_1_/DF1
                        // chain2---1---circle_i/reg_f_5_
                        // chain3---1---circle_i/reg_f_0_
0 edt_channel1 2 L H // chain1---2---circle_i/reg_x_2_/DF1
1 /x_out[3]          H L // PO
1 /x_out[2]          H L // PO
1 /x_out[1]          H L // PO
last_pattern_tested 1

```

Example 12

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```

read_patterns pat1
read_patterns pat2 -append
read_failures failure_file1
read_failures failure_file2 -append

```

Example 13

The following example loads the top-level *fail1.top* failure file.

```
read_failures fail1.top
```

Related Topics

[diagnose_failures](#)
[report_failures](#)
[set_diagnosis_options](#)

[write_failures](#)

read_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.

Usage

Path Delay Faults Usage:

```
read_fault_sites filename... [-Force | -Noforce]  
[-INSTance instance_name... | -MODule module_name...] [{> | >>} output_file_name]
```

Bridge Faults Usage:

```
read_fault_sites filename...  
[-INSTance instance_name... | -MODule module_name...]  
[-VERbose [All | Modules | Instances]] [-NAMES_ignored]  
{[-NOEQ] |  
 {[-Filter "{DISTance | PARAllel_run | WEIGHT} operator number"]...  
 [-Filter "TYPE {= | !=} type_identifier"]...  
 [-Filter "LAYER {= | !=} string"]...  
 [-SElect number[%]]...}} [{> | >>} output_file_name]
```

UDFM Faults Usage:

```
read_fault_sites filename [-VERbose]  
[-INSTance instance_name... | -MODule module_name...]
```

Description

Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.

Bridge Fault Specifics

A bridge fault definition file or output file from the Calibre query server can be loaded with this command. If a syntax error is encountered, the tool issues an error message and stops loading the bridge entries from the file. If a non-syntactical error occurs, an error displays but the file continues loading.

After bridge loading completes, the tool displays a summary of all errors and warnings encountered during the load. The summary lists how many bridges could not be mapped and why. For example, a bridge cannot be mapped if it relates to a location within a library cell.

Note

 If you define one or more filters using the -Filter switch, the command will load only those bridges that satisfy all specified filter criteria.

For more information on bridge faults and the bridge fault definition file, refer to “[Net Pair Identification with Calibre for Bridge Fault Test Patterns](#)” in the *Tessent Scan and ATPG User’s Manual*.

Path Delay Specifics

Before you use the tool to create ATPG patterns to detect path delay faults, first place the paths into a properly formatted file. Then use this command to load the file, and the add_faults or read_faults command to add the faults on the specified paths. You must also specify the path delay fault type with the set_fault_type command.

If you return to Setup mode with faults in the path delay fault list, the tool deletes the faults in that list and issues a warning message.

You can use multiple read_fault_sites commands and the results are additive. However, if you do not use the -Force switch, and one of the following conditions is met when the tool reads in the path data, the tool will issue an error message and terminate the command:

- The path name must not be the same as the name of an existing path.
- The first pin for a path must be a valid launch point. A valid launch point is a primary output of a scan cell state element, or a non-scan state element that satisfies the C1 clock rule.

If the path includes a clock or state element D-input pin, you must include the state element name in the path (or use the -Force switch). Fail to do so, and the tool will not resolve the path and report an error.

- The last pin for a path must be a valid capture point or a clock input of a scan cell. A valid capture point is a primary output or a data input of a cell that can capture data, whether scanned or not. Examples of invalid capture points would be a latch that is transparent or a memory element that cannot capture data due to a DRC violation, pin constraint, or cell constraint.
- Each pin must have unambiguous fan-in connectivity to the preceding pin, which must not tie to a constant logic value. If the pin fails to have a valid connection with the preceding pin, the tool generates an error and terminates the read_fault_sites command. However, if there is ambiguity in the connectivity, the tool selects a path between the pin and the preceding pin and generates a warning message. You can display the gates in the complete path using the Report Path command.
- Paths cannot propagate through RAM gates, ROM gates, or transparent latches.
- Paths cannot have edge ambiguity during any point in the path. An edge that propagates through XOR gates or the select lines of MUX gates can result in either a rising or falling edge at the gate outputs. You can use inversion parity to avoid edge ambiguity. If this check fails, the tool generates a warning, and you can assume that an edge on the pin is not inverted relative to the preceding pin.

- The condition statements in the path definition file must occur before the first pin statement and before the tool checks for valid pin names and values. The tool does not use the conditions to resolve edge or path ambiguities.

For more information about path delay faults and the path definition file, refer to “[Path Delay Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

UDFM Specifics

You use this command to load a UDFM definition file. If the tool encounters an error during loading, the tool stops the loading operation and displays a message. When the UDFM file is successfully loaded, the tool displays a summary similar to the following:

```
// UDFM definitions read: Total=1532, Cells=92, Modules=0, Instances=0,  
// No Match=476
```

The summary lists how many UDFM definitions were read from the file and how many of them have been added to the list of fault sites. In the above example, 1532 UDFM definitions are read from the file and 92 cells are added to the list of fault sites. The last value gives you the number of the UDFM definitions that did not match a cell or module in the design. You can use the -Verbose switch to report more detailed warning messages about the UDFM definitions that were not added to the list of fault sites.

For more information about UDFM, refer to “[Overview of Test Types and Fault Models](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- ***filename***
A required repeatable string that specifies the pathname of the path or bridge definition file or the UDFM file to load. This file is an ASCII file that defines the path delay or bridge fault sites or user-defined fault models for ATPG.
- **-Force**
An optional switch that specifies continued path reading after the first occurrence of an invalid path. If you do not specify this switch and the command encounters an invalid path, the command generates an error and terminates.
- **-Noforce**
An optional switch that specifies for the tool to stop path reading after the first occurrence of an invalid path. This is the default.
- **-INSTance *instance_name...***
An optional switch and repeatable string that causes each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify

the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

- **-MODule *module_name*...**

An optional switch and repeatable string that causes each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose actual module name matches the regular expression. The fault file will be loaded for each instance.

- **-VERbose**

An optional switch that enables the display of parsing errors and warnings while the path, bridge, or UDFM ASCII file is loaded.

For bridge entries only, this switch displays a summary of all errors and warnings once bridge entries are loaded. You can specify how these errors and warnings display with one of the following literals:

All — Displays all errors/warnings during loading and a summary report after loading.

Modules — Displays the number of bridges along with the names of the related modules that could not be mapped.

Instances — Displays the number of bridges along with the names of the related modules and instances that could not be mapped.

By default, only the summary report displays at the conclusion of the load operation.

- **-NAmes_ignored**

An optional switch that directs the tool to ignore all bridge names while loading. Use this option to load files having duplicate bridge names in the bridge definition or Calibre query server file, or to save memory.

- **-NOEQ**

An optional switch that prevents the loading of equivalent fault sites. This has the advantage of producing the smallest peak memory demand, as well as the smallest final memory demand, for loading.

- **-Filter {"{DISTance | PARAllel_run | WEIGHT} *operator number***

An optional repeatable switch, literal, operator and number quadruplet that filters which bridge entries are loaded based on a specified physical attribute of the bridge. A single blank space is required between each member of the quadruplet and the literal, operator, and number portion of the quadruplet must be enclosed in double quotes. The *number* must be a nonnegative number that applies to the specified attribute in the bridge entry. Attributes include:

DISTANCE — Real number that specifies the distance attribute in the bridge entry.

PARALLEL_run — Real number that specifies the parallel_run attribute in the bridge entry.

WEIGHT — Real number that specifies the weight attribute in the bridge entry.

The *operator* must be one of the following:

< less than

<= less than or equal to

= equal to

!= not equal to

>= greater than or equal to

> greater than

Due to rounding, it is not advisable to use “=” or “!=” for floating point numbers.

Note

 The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter “**TYPE {= | !=} type_identifier**”

An optional repeatable switch, literal, operator and literal quadruplet that filters which bridge entries are loaded based on the bridge type. A single blank space is required between each member of the quadruplet and the literal, operator, and literal portion of the quadruplet must be enclosed in double quotes. Select one of the following literals as the *type_identifier*:

S2S — Side-to-side

SW2S — Side-Wide-to-Side: same as S2S but at least one of the two signal lines is a wide metal line.

S2SOW — Side-to-Side-Over-Wide: same as S2S, but the bridge is located over a wide piece of metal in a layer below.

C2C — Corner-to-Corner

V2V — Via-to-Via: an S2S for vias.

VC2VC — Via-Corner-to-Via-Corner

EOL — End-of-Line: the end head of a line faces another metal line.

Note

 The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter “**LAYER {= | !=} string**”

An optional repeatable switch, literal, operator and string quadruplet that filters which bridge entries are loaded based on the layer attribute of the bridge. A single blank space is required between each member of the quadruplet and the literal, operator, and string portion

of the quadruplet must be enclosed in double quotes. The *string* must be the string value that specifies the layer attribute in the bridge entry.

Note

 The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -SElect *integer*[%]

An optional repeatable switch and positive integer pair that specifies to keep the *integer* best bridges of each bridge type. The tool determines the best based on the weight of each bridge after loading all fault locations in all files specified in the command line.

If you include the optional “%” character after the *integer*, it is interpreted as a percentage instead of an absolute number. For example, “-select 20%” directs the tool to retain the best 20% of all bridges after loading, whereas “-select 20” means to keep the twenty best bridges.

Note

 A weight attribute must be defined for each bridge in the file for this option to be meaningful. If a weight is not defined for a bridge, it is loaded regardless of this switch.

- > | >> *output_file_name*

An optional switch that writes the parsing errors, warnings and summary report to the specified file. Use this switch at the end of the command line. Options include:

- > Creates a new or overwrites an existing file.
- >> Appends the output to an existing file.

Examples

Example 1

The following path delay example sets up the tool to perform ATPG for the specified path delay faults in the path definition file, /user/design/pathfile.

First, you must set the fault type and read in the paths:

```
set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

```
report_faults -all
type  code  name_of_path
----- -----
//  Warning: No faults reported.
```

Next, add the faults on the paths contained in that file:

```
add_faults -all
report_faults -all
type  code  name_of_path
----- -----
0    UC    path0
1    UC    path0
```

Now you are ready to perform ATPG on the path delay faults:

create_patterns

The following is an example of a path definition file that contains one path:

```
PATH "path0" =
  PIN /I$6/Q + ;
  PIN /I$35/B0 + ;
  PIN /I$35/C0 + ;
  PIN /I$1/I$650/IN + ;
  PIN /I$1/I$650/OUT - ;
  PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge fault example sets up the tool to perform ATPG for the bridge fault sites specified in the bridge fault definition file, */user/design/bridgefile*.

First, you must set the fault type and load the bridge sites from the bridge fault definition file:

```
set_fault_type bridge
read_fault_sites /user/design/bridgefile

// 
// Load bridge entries from file "/user/design/bridgefile".
//
// 5 bridge entries were read from file "/user/design/bridgefile".
// 2 bridge entries were skipped due to unknown net name.
// 3 bridge entries were added to the fault site list.
```

report_fault_sites

```

BRIDGE {
    NET1 = "/G5";
    NET2 = "/G10";
    FAULTS = {-, -, -, -};
    NAME = "U7.BRIDGE";
}
BRIDGE {
    NET1 = "/G4";
    NET2 = "/G10";
    FAULTS = {-, -, -, -};
    NAME = "U5.BRIDGE";
}
BRIDGE {
    NET1 = "/G11";
    NET2 = "/G10";
    FAULTS = {-, -, -, -};
    NAME = "U0.BRIDGE";
}

```

Next, add faults on the sites in the fault site list:

```

add_faults -all
report_fault_sites

BRIDGE {
    NET1 = "/G5";
    NET2 = "/G10";
    FAULTS = {UC, UC, UC, UC};
    NAME = "U7.BRIDGE";
}
BRIDGE {
    NET1 = "/G4";
    NET2 = "/G10";
    FAULTS = {UC, UC, UC, UC};
    NAME = "U5.BRIDGE";
}
BRIDGE {
    NET1 = "/G11";
    NET2 = "/G10";
    FAULTS = {UC, UC, UC, UC};
    NAME = "U0.BRIDGE";
}

```

Now you are ready to perform ATPG on the bridge faults:

create_patterns

Example 3

The following is an example of a bridge fault definition file that contains one bridge site:

```
VERSION 1.1
BRIDGE {
    NET1="G173";
    NET2="G1296";
    NAME="My_layout.BRIDGE_TYPE1";
    DISTANCE=4.93827e-06 um;
    PARALLEL_run=6.075e-05 um;
    WEIGHT=95
    X_COORDINATE=152500;
    Y_COORDINATE=739500;
    LAYER="metal1";
    TYPE="S2S";
}
```

Example 4

The following example filters bridge entries based on the parallel_run attribute and loads only the bridge entries whose parallel_run attribute is between 7.5e-05 and 8e-05 um into the current fault definition list.

```
delete_fault_sites -all
read_fault_sites /user/design/bridgefile -filter "parallel_run > 7.5e-05" \
               -filter "parallel_run < 8e-05"

// 5 bridge entries were read from file "/user/design/bridgefile".
// 2 bridge entries were skipped due to unknown net name.
// 2 bridge entries were filtered out.
// 1 bridge entry was added to the fault site list.
```

Notice the tool loads only the bridge entries that meet the specified filter criteria, but does not otherwise utilize the attributes. To see the attributes, include the -Attribute switch when you report the fault sites.

Example 5

The following example loads all bridge entries from the *z_bridge_definitions* file and writes loading warnings and errors to the *loading_errors* file:

```
read_fault_sites z_bridge_definitions > loading_errors
// ... writing to file loading_errors
```

Example 6

In the following example, remember that the `read_fault_sites` does not actually add the fault to be targeted by ATPG, but only creates a named fault for referencing later. However, `create_patterns` automatically adds the loaded faults when issued.

run 1: You create faults via physical extraction for module alu. You complete the run and are satisfied with the block coverage.

```
read_fault_sites alu_bridges_file
create_patterns
```

run 2a: You then embed the alu design from run 1 in a large design as instance alu1. You want to reuse the fault file used in run 1 to define the faults within that particular instance of the new larger design.

```
read_fault_sites alu_bridges_file -instance alu1
```

You want all bridges loaded to be targeted, except for the one in the fault file named *bridge7*. Bridge fault sites must have names in the file and you cannot have used the *-names_ignored* switch, which throws away the names to save internal storage.

```
delete fault site alu1/bridge7 -name
create_patterns # automatically adds all nondeleted, loaded faults ...
```

run 2b: You embed the alu design 10 times as a different instance and want to reuse the fault file from run 1 to define the faults in each of the 10 instances of module alu.

```
read_fault_sites alu_bridges_file -module alu
```

run 2c: You embed the *alu* design as *inst1* and *inst2*; because no other instances begin with “nest”, you can use the * wildcard as shown:

```
read_fault_sites alu_bridges_file -inst inst*
```

In all of these runs, you can issue [create_patterns](#) after loading and deleting fault sites to obtain the desired set. The [create_patterns](#) command automatically adds faults at every loaded, undeleted fault site for ATPG to target.

Example 7

The following example loads two UDFM files into the current tool session and displays parsing errors and warnings while loading the second file:

```
read_fault_sites /home/design/c090_std/c090_std.udfm
read_fault_sites /home/design/c090_io/c090_io.udfm -verbose
```

```
// Warning: UDFM - File c090_io.udfm, line 31 - Cell definition u_and2 does not exist in the
// circuit.
```

Related Topics

[add_faults](#)
[delete_fault_sites](#)
[report_fault_sites](#)
[set_fault_type](#)
[write_fault_sites](#)

read_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Updates the current fault list with the faults contained in the specified fault file or from the Tessent Shell Database (TSDB).

Usage

```
read_faults [filename]
{[-INSTance instance_name... | -MODule module_name...] [-VERBose]
[-GRaybox] [-KEEP_Patterns]} [-SHow_warnings]
[-FAULT_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}]
[-MODE mode_name] [-PATtern_id pattern_id]
[-DESign_id design_id] [-SIlent]
```

Stuck/Transition/Toggle/Iddq/Path_delay specific options:

```
[-ALLOW_fault_sampling | -RETain | -Delete | -DELETE_Equivalent |
-Merge [-NO_Warnings] ]
[-DETection {Append | Replace}]
```

Bridge specific options::

```
[-RETain | -Merge]
```

UDFM specific options:

```
[-RETain | -Delete | -Merge] [-DETection {Append | Replace}]
```

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):

```
[-POWer_check {ON | Off}]
```

Description

The read_faults command affects the current fault population by either adding or removing faults that you specify in an external fault file or that were written to the TSDB. Because you must identify the faults before performing ATPG or fault simulation, this command is useful when you have a large number of faults to identify. The power-aware option loads faults based on the active power mode.

Provide the faults by specifying the fault filename or, if you used the [write_tsdb_data](#) command during pattern generation, specifying one of these switches, instead of the pattern filename, which eliminates the need to remember the location of the pattern file and allows the tool to locate the patterns in the TSDB:

- -mode
- -fault_type
- -pattern_id
- -design_id

For more information, refer to [logic_test_cores](#) in “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763.

The `read_faults` command supports the MTFI (Mentor Tessent Fault Information) format. For more information about MTFI, refer to the [Tessent Scan and ATPG User’s Manual](#).

Path Delay/Stuck/Transition/Toggle/Iddq Faults Specifics

By default, if you do not specify the `-Retain` switch, the tool initially places all the faults in the fault file into the uncontrolled (UC) fault class. It then analyzes and reclassifies them appropriately.

Note

 The filename specified cannot have fault information lines with comments appended to the end of the lines or fault information lines greater than six columns. If it does, the tool will not recognize the line properly and will not add the fault on that line to the fault list.

The format of the fault file data can be either in a three-, four-, or six-column standard format. Regardless of the format, the `read_faults` command uses only the information in the first, second, and last columns. You may also place comment lines in the file by starting the line with a double slash (//); the tool ignores comment lines. The file follows the format illustrated below:

```
stuck fault_class (cellname) (netname) (ignore) pin_pathname
```

- **stuck** — The first column must be the stuck-at value.
- **fault_class** — The second column must be the fault class value, but only if you use the `-Retain` option. For detailed information on fault classes, refer to the “[Fault Classes](#)” section in the [Tessent Scan and ATPG User’s Manual](#).
- **cellname** — The third column is the cell name enclosed in parenthesis. When present, this column indicates the type of cell in which the fault resides.
- **netname** — The fourth column is the net name enclosed in parenthesis. When present, this column indicates the net in which the fault resides.
- **ignore** — The fifth column is ignored by both tools.
- **pin_pathname** — The last column must be the pin pathname.

Here is an example of a short fault file that provides just the data the tool uses (as described in the first, second, and last bullets above):

```
// Short fault file example
1 UC /u510/u17/A0
1 UC /u510/u17/Y
0 UU /u510/u17/A1
1 UU /u510/u17/A1
```

Note

 When black boxes are present in the design, the `read_faults` command is unable to load the faults that are inside the black boxes. Instead, the tool discards those faults and issues warning messages. The result is fewer faults loaded into the tool.

Arguments

- *filename*

An optional string that specifies the name of the ASCII file containing the fault list to load. The command automatically detects when *filename* is an MTFI file. You can enter only one string.

This switch is used when you are reading faults from a file rather than the TSDB. If you used the [write_tsdb_data](#) command during pattern generation, the patterns are located in the TSDB. Specifying one of these switches eliminates the need to remember the location of the pattern file and allows the tool to locate the patterns in the TSDB: `-mode`, `-fault_type`, `-design_id`, and `-pattern_id`. The *filename* cannot be used if you specify one of these switches.

- `-INSTance instance_name...`

An optional switch and repeatable string that cause each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument, if the *filename* is specified. You can use the asterisk (*) and question mark (?) wildcards to specify the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

If this switch is specified without a *filename*, the tool uses the specified instance names to retrieve the module names of those instances to do module matching (see [set_module_matching_options](#)) against the *design_name* of the files in the TSDB and only read the fault files that match, applying each file to the corresponding instance of the matching module.

- `-MODule module_name...`

An optional switch and repeatable string that cause each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument, if the *filename* is specified. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose actual module name matches the regular expression. The fault file will be loaded for each instance.

If this switch is specified without a file name, the tool uses the specified module name to do module matching (see [set_module_matching_options](#)) against the *design_name* of the files in the TSDB and only read the fault files that match, applying each file to the corresponding instances of the matching module.

- **-VERBose**

An optional switch that applies only to *-Instance* and *-Module*. Prior to being embedded, top-level IO pins on an instance are faultable sites; after being embedded, they no longer are fault sites. In this case, a summary message states that these faults cannot be loaded due to no longer being a faultable site. If the -Verbose switch is issued, an individual message for each unloadable fault is also issued. By default, this switch is disabled.

- **-Graybox**

An optional switch that causes the instances specified with the *-Instance* or *-Module* switch to be treated as gray boxes. When you use the *-Graybox* switch, you must use either the *-Instance* or *-Module* switch.

Note



The *-graybox* switch cannot be combined with the *-retain*, *-delete*, or *-delete_equivalent* switches.

When the tool can translate the loading faults to an existing net, the *-Graybox* switch merges the faults to the internal fault list. When the tool cannot translate a loading fault to an existing net, the tool designates that fault as unlisted and retains its detection status.

Note that this option does not support reading multiple fault lists for the same instance or module. If your intent is to merge multiple fault lists for a given core, then you must perform the merge during the core-level ATPG run using the “*read_faults -merge*” command. Note also that you can only merge fault lists of a single fault type. You cannot, for example, merge fault lists for stuck and transition at the core level.

For more information about using the *-Graybox* switch and hierarchical fault accounting, refer to “[Support for Hierarchical Fault Accounting](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-DETection {Append | Replace}**

An optional switch and literal operand that specify the operation on the n-detection number of faults in the FaultList data block of the loading MTFI file.

The Append literal appends the n-detection number stored in the external MTFI file to the n-detection number of the corresponding fault. The append operation caps at the detection limit set by the [set_multiple_detection](#) command. When you use either the *-Merge* or *-Graybox* switch, the “-Detection Append” switch automatically applies. When you use the “-Detection Append” switch, you must also include either the *-Merge* or *-Graybox* switch.

The Replace literal replaces the n-detection number of the fault in the internal fault list with the n-detection value of the corresponding fault loaded from the MTFI file. When you use the “-Detection Replace” switch, you must also use the *-Retain* switch.

- **-KEEP_Patterns**

An optional switch that specifies that the tool preserves the current internal patterns while loading faults.

Note

 When this switch is set, the new faults loaded are not fault graded by the existing internal patterns, so the new fault status is not reflected by the internal pattern set.

- **-ALLOW_fault_sampling**

An optional switch that enables fault sampling when loading faults.

By default, the `read_faults` command ignores fault sampling rate data unless you specify this switch. If the fault sampling rate is other than 100%, and you do not specify the `-ALLOW_fault_sampling` switch, then the tool resets the sampling rate; for subsequent commands, you must re-specify the sampling rate.

Note

 The behavior for the `-Delete` or `-DELETE_equiv` switches remain unchanged irrespective of whether you use this switch or not.

- **-RETain**

An optional switch that specifies that the tool retains the fault class of each fault in the fault list.

- **-Delete**

An optional switch that specifies that the tool removes all the *filename* faults [delete_faults](#)

- **-DELETE_Equivalent**

An optional switch that specifies that the tool removes *filename* faults from the current fault population within the tool's session, and to remove all the faults that are equivalent to those in *filename*.

- **-Merge**

An optional switch that merges the loaded external faults with the internal fault list. This allows you to combine the fault lists from different ATPG sessions. For example, if you partition your design and run ATPG for each partition, you can get chip-level fault statistics by merging the partition fault lists. Also, for a single design or design partition, you can run ATPG once, then you can target more undetected faults by incrementally running ATPG using different settings and merging fault lists. For more information, see Example 2.

If a fault being loaded does not exist in the internal fault list, it will be added to the internal fault list.

If a fault being loaded already exists in the internal fault list, the fault class of the existing fault will be updated with the highest priority fault class. For example, if the fault being loaded has a DI class and the matching fault in the internal list has a PT class, then the class of the fault in the internal list will be updated to DI because DI has a higher priority than PT.

Fault classes are prioritized as follows:

DR > DS > DF > DI > TI > RE > BL > PT > PU > UO > UC > AU > UU

-NO_Warnings

An optional switch that suppresses warnings about conflicting untestable fault classes while merging fault lists. If a fault being loaded has a different untestable fault class than a matching fault in the internal fault list, the fault class defined in the internal list will be used.

- **-Power_check {ON | Off}**

An optional switch and literal that control the loading of faults in a power-off domain. The default is Off. This switch is applicable only after you have loaded a CPF or UPF file.

ON — A literal that loads the faults on a power-on domain and discards the faults in a power-off domain. Use this option to calculate the overall test coverage, including the faults that are not active in the current active power mode.

Off — A literal that allows all the faults to be loaded without checking the power mode.

- **-SHow_warnings**

An optional switch that allows the tool to report errors at each line of the file when reading faults. By default, the command ignores errors encountered while reading the fault list and summarizes the errors at the command's completion.

- **-fault_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}**

An optional switch and literal string that specify the fault type. The literal string fault type is one of these fault types:

- stuck
- iddq
- toggle
- transition
- path_delay
- bridge
- udfm

This switch is required if the *filename* is omitted.

- **-mode *mode_name***

An optional switch and string that specify the mode to read faults for. The mode name is captured, along with its type, in the Tesson Core Description (TCD) file, which is used for retargeting or core mapping.

If the -mode switch is not specified, the tool treats it as unspecified and searches the TSDB for one single file with a mode that matches the specified criteria for the command. The tool issues an error and prompts for additional criteria (-mode) if multiple files with different mode names match the specified criteria.

This switch cannot be used if you specify the filename.

For more information on modes, refer to “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan an ATPG User’s Manual*.

- **-pattern_id *pattern_id***

An optional switch and string that specify the *pattern_id* used when saving the faults to the TSDB directory. The *pattern_id* differentiates the multiple pattern sets for a given mode name of a given design view.

If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the desired pattern file. If one of the matching files does not have a pattern ID and you want to select it, specify -pattern_id with the empty (“”) string.

This switch cannot be specified if you use the filename.

- **-design_id *design_id***

An optional switch and string that specify the *design_id* used when creating the *logic_test_cores* TSDB sub-directory. When it is used, the tool uses the specified ID. If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

The *design_id* defines the design view used and is typically used to differentiate the pre-layout vs. post-layout design views so you can create post-layout patterns without losing pre-layout copies.

Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by [read_design](#) when reading a design. It also is set after elaboration on the current design to the value returned by the “[get_context -design_id](#)” command when the value is not null (when the design is loaded with [read_verilog](#) command rather than [read_design](#) command).

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the desired fault file. If one of the matching files does not have a design ID and you want to select it, specify -design_id with the empty (“”) string

This switch cannot be specified if you use the filename.

- **-Silent**

An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Examples

Example 1

The following example modifies the current fault population in the tool with the contents of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

The example reads in the detected faults from the fault file, restricting the tool to generating patterns for only undetected faults.

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge
report_faults -all
create_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage:

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault -merge
reset_au_faults -reclassify
create_patterns
report_statistics
write_faults modes_1_and_2.fault -replace
```

Example 3

The following example illustrates the reuse of a faults file in a case that needs the -instance switch.

Tip

i As a reminder, the `read_faults` command automatically adds the faults as well, assuming that they reference faultable sites. The `read_faults` command should never be followed by the “`add_faults -All`” command; that sequence of commands is equivalent to only issuing “`add_faults -All`” and is never what you intend. If you do not issue the `add_faults` command and only issue the `read_faults` command, `create_patterns` targets only the faults loaded, which is the desired result.

run 1: The user has an initial design that will eventually be part of a larger design. This initial design has a special subset of all stuck-at faults that are created via nofaulting and/or deleting faults. This fault set needs to be reused when this initial design is reused in the larger design. In this run, we assume that the user works and obtains adequate fault coverage. In order to be able to reuse the faults, the faults are written out to a file *stuck_file*.

run 2: The user has embedded the initial design from run 1 into a larger design; the instance of the initial design is named *alu1*. The user now wants to reuse the *stuck_file* fault file (created in run 1) to define the faults within that particular instance (*alu1*) of the new larger design.

Assume that a PI of the alu is named *inA*. Note that this will no longer be a faultable site once embedded (no longer a top level IO pin, or on a model boundary).

```
read_fault_sites stuck_file -instance alu1 -verbose
// Loading faults for instance alu1.
// Note: Original fault /inA is no longer a faultable site.
// Could not load alu1/inA
// Note: 1 fault not loaded because no longer a faultable site.
```

The first note in the output is omitted if -Verbose is not issued. The summary line in the second note is always issued to indicate how many were not loaded and added due to no longer being a faultable site, unless none were rejected for that reason.

Assume that a fault in the file *dff_32/Q* was loaded and therefore added, but the user wants to delete that fault. Because it is inside *alu1* once loaded, that must also be used in the reference to unambiguously identify the fault.

```
delete_faults alu1/dff_32/Q -s 0
```

Example 4

The following example modifies the current fault population by merging in the fault classifications of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge -power_check on
```

Example 5

The following example reads the faults for instance ELT_edt_aux_I1 from the .flt file, and treats that instance as a graybox:

```
read_faults det1_all.flt -instance ELT_edt_aux_I1 -graybox -verbose
```

Example 6

The following example updates the faults in the current fault list with the contents of the external UDFM file *c090_std_io.udfm* and displays parsing errors and warnings:

```
read_faults /home/design/c090_io/c090_std.io.udfm -verbose
// Warning: UDFM - File c090_io.udfm, line 31 - Cell definition
// u_and2 does not exist in the circuit.
```

Example 7

This example demonstrates reading faults from the TSDB created with the `write_tsdb_data` command. The `read_faults` command specifies the fault type, instance, and mode.

```
read_faults -fault_type stuck -instance my_instance -mode internal -silent
```

Example 8

This example demonstrates reading faults from the TSDB created with the `write_tsdb_data` command. The `read_faults` command specifies the fault type, instance, mode, and pattern ID.

```
read_faults -fault_type stuck -instance my_instance -pattern_id pat_id1 -mode internal -silent
```

In this case, multiple fault files match the specified criteria. The tool reports this message:

```
// Error: There are multiple fault files found in the opened TSDBs for
// instance 'my_instance' (core: 'my_core') with fault type 'stuck' and
// with pattern ID 'pat_id1' and with mode 'internal':
//     Fault file 'my_core_internal_stuck.faults.gz' in
//     TSDB './tsdb_outdir' with design ID ''
//     Fault file 'my_core_internal_stuck.faults.gz' in
//     TSDB './tsdb_outdir2' with design ID 'design_id'
//
// Please specify the design ID with the -design_id switch.
```

Example 9

This example demonstrates how you resolve the error produced in [Example 8](#). As stated in the error message, the specified criteria matches two fault files with different design IDs, design ID '' and 'design_id'.

```
read_faults -fault_type stuck -instance my_instance -pattern_id pat_id1 \
-d design_id "" -mode internal -silent
```

The design ID with the empty string is specified.

Example 10

This example shows the reporting of the auto-loaded fault file when you do not specify the -silent switch.

```
read_faults -fault_type stuck
```

```
// Reading fault file ./tsdb_outdir/logic_test_cores/
small_design.logic_test_core/small_design.atpg_mode_internal/
small_design_internal_stuck.faults.gz
```

Related Topics

[add_faults](#)
[delete_faults](#)
[read_cpf](#)
[read_upf](#)
[report_faults](#)
[report_testbench_simulation_options](#)
[set_attribute_value](#)
[set_fault_mode](#)
[set_fault_sampling](#)
[set_fault_type](#)

read_flat_model

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Loads a flat model into the tool from a specified filename or from the Tesson Shell database (TSDB).

Usage

```
read_flat_model [design_path] [-MODE mode_name] [-DESIGN_Id design_id]
[-DESIGN_Name design_name] [-enable_full_introspection {on | off}]
[-SILENT]
```

Description

Loads a flat model into the tool. The tool can only read in one flat model at a time.

If an ICL description is required to process the test_setup procedure or the test_end procedure stored in the flat model, then the appropriate ICL files must be read prior to the flat model.

Using TSDB to Locate the Flat Model

If you used the [write_tsdb_data](#) command, it is not necessary to supply the *design_path* because the information can be located in the TSDB using at least one of these switches:

- -mode
- -design_id
- -design_name

Using the TSDB and specifying one of these switches eliminates the need to remember the location of the flat model file.

Tcl Variables Stored in the Flat Model

When reading in a flat model, the tool only restores a Tcl variable if it is not already set in the tool, that is, existing Tcl variables are not overwritten with the exception of an existing “tesson_user_arg” variable that has no value. For array variables, the tool checks each individual array element.

Reading Password Protected Flat Model Files

When reading in a flat model that was saved using the “[write_flat_model](#) -password *password*” command, the tool requires you to enter the password that was provided with the switch when the model was saved. For batch runs, you can set the MGC_DFT_FLAT_MODEL_PASSWORD environment variable to the specified password; the tool obtains the password from the environment variable instead of querying for it. If the password setting of this variable is incorrect, however, the tool will revert to querying you for it.

Arguments

- *design_path*

A optional string that specifies the pathname of a flat model. If it is not specified, the tool automatically searches for a matching flat model file in the opened TSDB for the current design, if one of the following switches is used: -mode, -design_id, and -design_name.

The automation for `read_flat_model` is available for the top-level design only.

- **-MODE** *mode_name*

An optional switch and string that specify the mode for which the flat model is read. The mode name is captured, along with its type, in the Tesson Core Description (TCD) file, which is used for retargeting or core mapping. See “[Tesson Shell Data Base \(TSDB\)](#)” on page 3763.

If the -mode switch is not specified, the tool treats it as unspecified and searches the TSDB for one single file with a mode that matches the specified criteria for the command. The tool issues an error and prompts for additional criteria (-mode) if multiple files with different mode names match the specified criteria.

This switch cannot be used with *design_path*.

For more information on modes, refer to “[Core Mapping for ATPG Process Overview](#)” in the *Tesson Scan an ATPG User’s Manual*.

- **-DESIGN_Id** *design_id*

An optional switch and string that specify the *design_id* used when creating the *logic_test_cores* TSDB sub-directory. When -design_id is used, the tool uses the specified ID. If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

The *design_id* defines the design view used and is typically used to differentiate the pre-layout vs. post-layout design views so you can create post-layout patterns without losing pre-layout copies.

Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by [read_design](#) when reading a design. It also is set after elaboration on the current design to the value returned by the “[get_context -design_id](#)” command when the value is not null (when the design is loaded with [read_verilog](#) command rather than [read_design](#) command).

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the desired flat model. If one of the matching files does not have a design ID and you want to select it, specify -design_id with the empty (“”) string

This switch cannot be used with *design_path*.

- **-DESIGN_Name** *design_name*

An optional switch and string that specify the design name for which the flat model should be read. The tool uses the specified design name to do module matching (see [set_module_matching_options](#)) against the *design_name* of the file and only reads the flat model file in the TSDB that matches. This can be used when the current design is not set.

This switch cannot be used with *design_path*.

- **-enable_full_introspection** on | off

An optional switch and literal that enable full introspection of the flat model. The default is on, which means the introspection commands (for example, get_modules, get_instances and so on) can be used.

This switch is disabled in the scan_diagnosis context.

For flat models created with pre-v2015.3 tool versions, full introspection may not be enabled.

- **-Silent**

An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Examples

Example 1

The following example loads a flat model:

```
read_flat_model /u/my_account/designs/model1
```

Example 2

This example demonstrates reading a flat model from a TSDB created with the write_tsdb_data command. The read_flat_model command specifies the design name.

```
read_flat_model -design_name small_design -silent
```

```
// Error: There are multiple flat model files found in the opened TSDB
// for the current design with design name 'small_design':
//     Flat model file 'small_design_internal.flat.gz' in
//         TSDB './tsdb_outdir' with mode 'internal'
//     Flat model file 'small_design_my_mode.flat.gz' in
//         TSDB './tsdb_outdir' with mode 'my_mode'
//
// Please specify the mode with the -mode switch.
```

In this case, the tool reports that it finds multiple flat model files in the TSDB for the current design with the design name small_design. The files have different mode names: “internal” and “my_mode”

Example 3

This example demonstrates how to resolve the error reported in [Example 2](#). The message asks for the mode to be specified with the -mode switch. In this case, mode “internal” is specified.

```
read_flat_model -design_name small_design -mode internal -silent
```

Example 4

This example demonstrates reading the flat model from a TSDB created with the write_tsdb_data command. The read_flat_model command specifies the design ID.

```
read_flat_model -design_name small_design -design_id my_design_id -silent
```

Example 5

This example demonstrates reporting the auto-loaded flat model when you do not specify the -silent switch.

```
read_flat_model -design_name small_design
```

```
// Reading flat model file ./tsdb_outdir/logic_test_cores/
small_design.logic_test_core/small_design.atpg_mode_internal/
small_design_internal.flat.gz
```

Related Topics

[delete_design](#)

[read_verilog](#)

[set_current_design](#)

[set_design_macros](#)

[delete_cell_library](#)

[write_flat_model](#)

read_icl

Context: all contexts

Mode: setup

Reads ICL files into the internal ICL database.

Usage

```
read_icl file_name_list [-bsdl_file bsdl_file] [-force]
```

Description

Reads ICL files into the internal ICL database.

Note

 If you want to read the ICL file located in one of the [dft_inserted_designs](#) directories, use the [read_design](#) command, which has a -design_identifier *design_identifier* switch-value pair. You only need to specify this option when the design is not already elaborated and the [Tessent Shell Data Base \(TSDB\)](#) contains multiple [dft_inserted_designs](#) views for the *design_name* you want to read in. Use the -no_hdl switch to only read the ICL view and not overwrite the design view.

Arguments

- *file_name_list*

A required list of strings that specifies the names of the ICL files to load. The names of the files can contain wildcards. You can define the list of files to read into a Tcl list and use it to load the files.

If you have used the command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- -bsdl_file *bsdl_file*

An optional value pair that defines the name of a BSDL file. The option is only needed and used when reading an ICL module containing an AccessLink of STD_1149_1_2001 element. The BSDL file must define an “Entity *entity_name* is” clause where *entity_name* is the one specified in the BSDLEntity property of the [AccessLink](#) wrapper. Refer to the Examples section of [AccessLink](#) for an example of the -bsdl_file usage.

- -force

An optional switch that suppresses the normal error message that is generated when the read_icl command is issued after the ICL elaboration has happened. With the -force option, the ICL elaboration tree is discarded instead and must be recreated afterward using the set_current_design command.

Return Values

None

Examples

In the following example, the first read_icl command reads a set of ICL files specified as multiple strings separated by spaces and surrounded by braces. The second read_icl command reads all files matched by the glob command. The third read_icl command reads all filenames contained within the Tcl list icl_list. In all of these cases, undefined ICL modules found within the modules inside the read ICL files are searched for in the specified icl_src directory.

```
set_design_source -format icl -y icl_src -extension icl
read_icl {chip.icl tdr.icl tap.icl block1.icl block2.icl}
read_icl [glob icl/*.icl]
set icl_list [list chip.icl tdr.icl tap.icl block1.icl block2.icl]
read_icl $icl_list
```

Related Topics

[delete_icl_modules](#)

read_lef

Context: dft, patterns

Mode: all modes

Prerequisites: LEF file

Reads in the LEF file so that the tool can extract the dimensions of the module.

Usage

`read_lef lef_file_name`

Description

This command reads in a LEF file so that the tool can extract the lower left and upper right coordinates of the rectangular boundary of the module and store them as attributes of the module. This information is useful when combined with the information extracted using the [read_def](#) command to precisely localize the module.

The tool stores the (x,y) coordinates of each instance in the following coordinate attributes.

- `def_xl_coordinate` — Lower left x coordinate
- `def_yl_coordinate` — Lower left y coordinate
- `def_xh_coordinate` — Upper right x coordinate
- `def_yh_coordinate` — Upper right y coordinate

All of the coordinates are floating point values in microns (um).

Arguments

- `lef_file_name`

A required string that specifies the name of the LEF file.

Related Topics

[read_def](#)

read_liberty

Context: unspecified, all contexts

Mode: setup

A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules.

Usage

`read_liberty liberty_filename`

Description

A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules. For more information, see “[Create Tessent Insertion Attributes Using Liberty](#).”

The cell attributes gathered from read_liberty are populated onto the read_cell_library model of the same name. This occurs when the populated cell information is needed, which is when the first `write_cell_library`, `get_dft_cell`, or `set_current_design` command is issued. It is advised that you make a separate read_liberty, read_cell_library, write_cell_library run as in the example below, then examine any Warnings and other messages produced, and possibly even edit the cell library output by that command (for example, to move a scan_equivalent to the front of the scan_equivalents list because only the front scan cell is used to replace a nonscan cell during test insertion). If manual edits are required, it is advisable to reiterate read_cell_library and write_cell_library, to ensure the manual edits were done correctly. For iterations, read_liberty should be omitted, because the cell attribute information from that command is already populated onto the models/cells written out by write_cell_library.

Arguments

- *liberty_filename*

The name of a Liberty cell attributes file.

Examples

```
read_cell_library tessent_cell.tcellib
read_liberty liberty.lib
write_cell_library tessent_cell.tcellib
```

read_modelfile

Context: all contexts

Mode: setup

Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.

Usage

```
read_modelfile {modelfile_name | -Delete} RAM/ROM_instance_name
```

Description

Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.

The read_modelfile command sets the initial memory states of a RAM or ROM gate using the data that you provide in a modelfile. You can create a modelfile from within the library cell or by using the write_modelfile command. The modelfile must contain initialization data that is in the Mentor Graphics modelfile format.

You specify the RAM or ROM gate that you want to initialize by using its instance name. An error condition occurs if the instance contains multiple RAM/ROM gates. When you issue the command, the flat model may not be available, so the tool checks for the correctness of the instance name and the modelfile name during rules checking.

You may also initialize memory states of a RAM or ROM gate by specifying the modelfile from within the RAM or ROM model description. To do so, use the init_file attribute. When using the init_file attribute, you must specify the full pathname to the initialization file if the file is not located in the directory from where you invoked the tool. For more information about modeling RAMs and ROMs and the init_file attribute, refer to the “[RAM and ROM](#)” subsection of the *Tessent Cell Library Manual*.

Modelfile Format

A Mentor Graphics modelfile contains addresses and data. You must present the addresses in hexadecimal or binary format. You can specify a range of addresses such as 0-1f. An address range may contain an asterisk (*) wildcard character. For example, to specify that you want all addresses set to a hexadecimal F, use “* / f;”. You cannot use an X in an address.

To add comments to a RAM or ROM initialization file, precede the comment text with two forward slashes (//). For example:

```
0/3;  
// here is a comment  
1/2;
```

You can present the data in either binary or hexadecimal format; the default is hexadecimal. To specify data in binary format, you must add a ‘%’ to the beginning of the data values. If you use an X within hexadecimal data, all four bits that it represents are Xs. Therefore, to set a single bit to X, use the binary format.

The following two examples are equivalent. The first example shows both an address and its associated data in hexadecimal. The second example shows the same address and data, but the data is now shown in binary.

```
ABCD / 123X;  
ABCD /%000100100011XXXX;
```

The following is an example of what an initialization file may look like (range 0-1f):

```
0/ a;  
1-f / 5;  
10 / 1a;  
11-1f / a;
```

Assuming that the ROM has 5 data bits per word, the least that can fit the largest data of “1a”, following this example, the contents of the memory would be:

Word (MS-LS)	/	Data (MS-LS)	
00000	/	01010	// Created by "0/a;"
00001	/	00101	// Created by "1-f/5;"
00010	/	0101	
00011	/	00101	...
...			
01111	/	00101	// Created by "1-f/5;"
10000	/	11010	// Created by "10/1a;"
10001	/	01010	// Created by "11-1f/a;"
10010	/	01010	...
...			
11111	/	01010	// Created by "11-1f/a;"

You can use an asterisk (*) for an address range. For example, you could rewrite the previous initialization file as shown here. In this example, the first line assigns the data value “a” to the full address range 0-1f. The subsequent lines overwrite the “a” data value with the new data values for the specified addresses.

```
* / a;  
1-f / 5;  
10 / 1a;
```

Making the same assumption as the previous example, following this example, the contents of the memory would be:

```
Word (MS-LS) / Data (MS-LS)
00000    / 01010 // Created by "*/a;" 
00001    / 00101 // Created by "1-f/5"; (replaces 'a' from */a)
00010    / 00101
00011    / 00101 ...
...
01111    / 00101 // Created by "1-f/5"; (replaces 'a' from */a)
10000    / 11010 // Created by "10/1a"; (replaces 'a' from */a)
10001    / 01010 // Created by "*/a;" 
10010    / 01010 ...
...
11111    / 01010 // Caused by "*/a;"
```

Arguments

- *modelfile_name*

A string that specifies the pathname to and filename of the modelfile containing the RAM or ROM initialization data in Mentor Graphics modelfile format.

- **-Delete**

A switch that deletes the modelfile initialization assignment you made previously for the specified RAM or ROM gate instance. Use this switch to undo an incorrectly specified modelfile or instance name (due, for example, to a misspelling or a non-existent modelfile or instance).

Note

 If you misspelled the RAM/ROM_instance_name when you first assigned the modelfile, you need to use the same spelling when you delete the assignment. After you remove the incorrect modelfile initialization assignment using the -Delete switch, you can re-issue the read_modelfile command using the corrected modelfile or instance name.

- *RAM/ROM_instance_name*

A required string that specifies the instance name of the RAM or ROM gate that you want to initialize, or for which you wish to delete a previous modelfile initialization assignment.

Examples

Example 1

The following example initializes the memory states of a RAM gate, then performs an ATPG run:

```
read_modelfile model.ram /p2.ram
set_system_mode analysis
create_patterns
```

If the memory primitive specified in the read_modelfile command did not exist, then the set_system_mode analysis command would result in an error message similar to the following:

```
// Error : /p2.ram given in an earlier read_modelfile command is NOT a  
// RAM or ROM primitive.
```

Example 2

The next example fixes the error by using the -Delete switch to remove the incorrect entry from the tool's internal list of RAMs and ROMs to be initialized. The example then re-issues the original command using the correct instance name (assumed to be /p1.ram):

```
read_modelfile -delete /p2.ram  
read_modelfile model.ram /p1.ram
```

Related Topics

[create_initialization_patterns](#)

[write_modelfile](#)

[set_ram_initialization](#)

read_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis,
patterns -scan_retargeting, patterns -failure_mapping

Mode: analysis

Loads the external pattern set from the specified file or from the Tesson Shell database (TSDB).

Usage

```
read_patterns [filename] [-append]
  [-fault_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}]
  [-mode mode_name] [-pattern_id pattern_id]
  [-design_id design_id] [-SIlent]
```

For patterns -scan_retargeting:

```
[-modules module_objects | -instances instance_objects]
```

All other contexts' specific options:

```
[-mask_file mask_filename [-maskfile_pattern_offset number]]
  [-preserve_masked_observe_points]
```

Description

For pattern retargeting, this command extracts the core name and mode from the pattern file and automatically associates the patterns with the correct core and mode.

This command appends to or replaces the external pattern set, and optionally provides additional observation masking information.

Provide the patterns by specifying the pattern filename, or if you used the [write_tsdb_data](#) command during pattern generation, specifying one of these switches, instead of the pattern filename, eliminates the need to remember the location of the pattern file and allows the tool to locate the patterns in the TSDB:

- -modules
- -instances
- -mode
- -fault_type
- -pattern_id
- -design_id

If `read_patterns` is issued in the “patterns -scan retargeting” context without a filename, the tool searches for pattern files for all of the instantiated TCD cores (added with `add_core_instances`), with a *design_name* equal to the core module name of those same cores and with a corresponding mode.

If `read_patterns` is issued in the “patterns -scan” context without a filename, the tool searches for a pattern file for the current design. The tool infers the core name from the current design’s module name by performing module matching (see [set_module_matching_options](#)) and treats the mode name as unspecified. The tool issues an error and prompts for additional criteria if multiple files with different mode names match the specified criteria. See [Example 5](#).

For more information, refer to [logic_test_cores](#) in the [Tessent Shell Data Base \(TSDB\)](#) section.

Arguments

- *filename*

A string that specifies a file containing the patterns to be read. It is used when you are reading patterns from a file rather than the TSDB.

This file is typically created with the [write_patterns](#) command. The patterns can be in ASCII, binary, pattern database (PatDB), STIL, or WGL test pattern format.

The tool obtains masking information from the patterns read in, applying masks wherever Xs appear in a pattern’s unload and measure_po values. Even when known values are determined after simulation, the tool continues to mask those locations.

If you used the [write_tsdb_data](#) command during pattern generation, the patterns are located in the TSDB. Specifying one of these switches eliminates the need to remember the location of the pattern file and allows the tool to locate the patterns in the TSDB: -mode, -fault_type, -design_id, -pattern_id, -instances, and -modules.

The filename cannot be used if you specify any of these switches: -mode, -fault_type, -design_id, and -pattern_id, -instances, and -modules.,

- `-append`

An optional switch that appends the new patterns on the existing external pattern set. By default, the command deletes the existing external pattern set and replaces with the new pattern set.

- `-mask_file mask_filename`

An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns, use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

The tool preserves existing masked observation points (scan cell or primary output) in external patterns loaded into the tool when a `mask_file` is specified.

This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using “`write_patterns -external`”, you will not see the masking information from the

mask file reflected in the saved patterns; the saved patterns will be exactly the same as the original patterns.

To save a pattern set that incorporates the masking information, use the [simulate_patterns](#) command to perform a good machine simulation of the external pattern set you specified with “read_patterns -preserve_masked_observe_points”. Then use the [write_patterns](#) command without the -External switch to write the resulting internal pattern set to a file.

The Mask File

- The *mask_filename* specifies a mask file, a text file you create that lists the patterns and locations you want masked.

Use the statements and syntax described in [Table 5-3](#) to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

The mask file must be in a pattern based format. Most ATE’s return accumulated cycle-based failure data. Use the [read_failures](#) command to convert cycle-based failure data to pattern based data for the mask file.

Table 5-3. Mask File Statements

Statement	Usage Rules
<location>	<p>Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows:</p> <ul style="list-style-type: none"> • If “type mask” or “type observe” precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name>

Table 5-3. Mask File Statements (cont.)

Statement	Usage Rules
type <value>	<p>Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values:</p> <ul style="list-style-type: none"> • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for <i>all</i> patterns. Overrides other type statements that cover the same location. <p>Use as many type statements as you like. By default, the tools assume “type mask” for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.</p>

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
// pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
// "type observe" for cell 3 of chain 5 for pattern 40, and
// subsequent "type observe" for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
// pattern 65.
65 chain5 99
```

- **-maskfile_pattern_offset *number***

An optional switch and integer that specify the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked. For more information, refer to [Example 2](#).

- **-preserve_masked_observe_points**

An optional switch that allows you to preserve existing masked observation points (wherever Xs appear in a pattern's unload and measure_po values) in external patterns loaded into the tool. When you use this switch to preserve existing masked observation points in the external patterns, the internal patterns stored after simulating the external patterns preserve these masked observation points. You can optionally use -mask_file to specify additional observation points to be masked.

- **-modules module_objects**

An optional switch and module object spec (an object name, a Tcl list of object names, or a collection of objects), only available for use in “patterns -scan_retargeting”, that specify modules added as core instances that the tool will associate with the patterns. When used, the tool searches the TSDB for pattern files for the cores of the specified modules. The core and mode names are inferred from the core instantiations (see [report_core_instances](#)).

This switch cannot be used if the -instances switch or *filename* is specified.

- **-instances instance_objects**

An optional switch and instance object spec (an object name, a Tcl list of object names, or a collection of objects), that specify instances added as core instances that the tool will associate with the patterns. When used, the tool searches the TSDB for pattern files for the cores of the specified instances. The core and mode names are inferred from the core instantiations (see [report_core_instances](#)).

This switch is only available for use in “patterns -scan_retargeting” and cannot be used if the -modules switch or *filename* is specified.

- **-fault_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}**

An optional switch and literal string that specify the fault type. The literal string fault type is one of these fault types:

- stuck
- iddq
- toggle
- transition
- path_delay
- bridge
- udfm

This switch is required if *filename* is omitted and cannot be used if the *filename* is specified.

- **-mode mode_name**

An optional switch and string that specify the mode to read the patterns for. The mode name is captured, along with its type, in the Tessent Core Description (TCD) file, which is used for retargeting or core mapping.

If -mode is not specified in non-retargeting flows, the tool treats it as unspecified and searches the TSDB for a mode that matches the specified criteria. If the tool finds an instance instantiated with multiple modes and a mode was not specified, it will report an error and prompt for a mode to be specified. See [Example 6](#).

For retargeting flows, the mode is inferred from the core instances added by the [add_core_instances](#) command. If -modules is specified, it uses the instances of the specified modules. If -instances is specified, it uses those instances.

You cannot use this switch if you specify the filename.

For more information on modes, refer to “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan an ATPG User’s Manual*.

- **-pattern_id pattern_id**

An optional switch and string that specify the *pattern_id* that was used when saving the patterns to the TSDB directory. The *pattern_id* is used to differentiate the multiple pattern sets for a given mode name of a given design view.

If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the desired pattern file. If one of the matching files does not have a pattern ID and you want to select it, specify -pattern_id with the empty (“”) string. See [Example 7](#).

You cannot use this switch if you specify the filename.

- **-design_id design_id**

An optional switch and string that specify the *design_id* that was used when creating the *logic_test_cores* TSDB sub-directory. When it is used, the tool uses the specified ID. When it is unspecified, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

The *design_id* defines the design view used and is typically used to differentiate the pre-layout versus post-layout design views so you can create post-layout patterns without losing the pre-layout copies.

Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by [read_design](#) when reading a design. It also is set after elaboration on the current design to the value returned by the “[get_context -design_id](#)” command when the value is not null (when the design is loaded with [read_verilog](#) command rather than [read_design](#) command).

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the desired pattern file. See [Example 4](#). If one of the matching files

does not have a design ID and you want to select it, specify -design_id with the empty ("") string.

You cannot use this switch if you specify the filename.

- **-Silent**

An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Examples

Example 1

The following example demonstrates how to perform fault grading for stuck-at faults using the transition pattern set created previously. Next, the example shows how to perform top-up ATPG.

```
set_context patterns -scan
set_system_mode analysis
set_fault_type stuck
add_faults -all
read_patterns pattern_transition
simulate_patterns -store_pattern none
create_patterns
write_patterns pattern_topup.stil -stil
```

Example 2

The following example demonstrates the use of the -maskfile_pattern_offset switch:

Consider the following mask file, named *mask_data.file*:

```
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
```

The following command reads the patterns from a STIL file, reads the above mask file, applying an offset of 400 in the mask file:

```
read_patterns pats_400_to_599.stil -mask_file mask_data.file -maskfile_pattern_offset 400
simulate_patterns -store_patterns all
write_patterns pats_400_to_599_masked.stil -stil
```

The result is that patterns 420 and 425 from the STIL file are masked.

Example 3

The “[Retargeting Example](#)” in the *Tessent Scan and ATPG User’s Manual* demonstrates the use of this command in a design context.

Example 4

The TSDB *logic_test_cores* directory structure is shown in Figure [5-2](#).

Figure 5-2. TSDB logic_test_cores Directory Structure

This example and [Example 5](#) have this TSDB *logic_test_cores* directory:

```

tsdb_outdir
  └── logic_test_cores
    └── core1_prelayout.logic_test_core
      └── core1.atpg_mode_internal
        └── core1_internal_stuck_patdId1.patdb
    └── core1_postlayout.logic_test_core
      └── core1.atpg_mode_internal
        └── core1_internal_stuck_patdId1.patdb
      └── core1.atpg_mode_special_mode
        └── core1_special_mode_stuck_patdId1.patdb
  
```

There are multiple pattern files, with different *design_ids* that match the specified criteria by the *read_patterns* command. The tool reports an error and prompts for additional criteria:

read_patterns –modules core1 –fault_type stuck -mode internal -silent

```

// Error: There are multiple pattern files found in the opened TSDBs
// for module 'core1' with fault type 'stuck' and with mode 'internal':
//   Pattern file 'core1_internal_stuck_patdId1.patdb' in
//     TSDB './tsdb_outdir' with design ID 'prelayout'
//   Pattern file 'core1_internal_stuck_patdId1.patdb' in
//     TSDB './tsdb_outdir' with design ID 'postlayout'
//
// Please specify the design ID with the -design_id switch.
  
```

To correct this error, specify a *design_id*:

```
read_patterns -modules core1 -fault_type stuck -mode internal -design_id postlayout -silent
```

There is a similar error message for multiple pattern files with different pattern IDs.

Example 5

The TSDB directory structure for this example is shown in [Example 4](#). This example demonstrates multiple pattern files with different mode names.

```
read_patterns -fault_type stuck -design_id postlayout -silent
```

```
// Error: There are multiple pattern files found in the opened TSDBs for
// the current design with fault type 'stuck' and with design ID
// 'postlayout':
//      Pattern file 'core1_internal_stuck_patdId1.patdb' in
//      TSDB './tsdb_outdir' with mode 'internal'
//      Pattern file 'core1_special_mode_stuck_patdId1.patdb' in
//      TSDB './tsdb_outdir' with mode 'special_mode'
//
// Please specify the mode with the -mode switch.
```

To correct this error, specify a mode:

```
read_patterns -fault_type stuck -design_id postlayout -mode internal -silent
```

Example 6

In this example, -mode was not specified and the tool finds an instance that has been instantiated with multiple modes:

```
read_patterns -instances {piccpu_inst} -fault_type stuck -silent
```

```
// Error: Instance 'piccpu_inst' of module 'piccpu' (core: 'piccpu_core')
// has been instantiated with modes 'internal1' and 'internal2'.
// Please specify the mode with the -mode switch.
```

To correct this error, specify a mode:

```
read_patterns -instances {piccpu_inst} -fault_type stuck -mode internal1 -silent
```

Example 7

In this example, there are multiple pattern files in the opened TSDB that match the specified criteria, but with different pattern IDs.

```
read_patterns -instances [get_instances core_1] -fault_type stuck -silent
```

```
// Error: There are multiple pattern files found in the opened TSDB for
// instance 'core_1' (core: 'piccpu_maxlen16_1', mode: 'internal') with
// fault type 'stuck':
//     Pattern file 'piccpu_maxlen16_1_internal_stuck.patdb' in
//         TSDB './tsdb_outdir' with pattern ID ''
//     Pattern file 'piccpu_maxlen16_1_internal_stuck_id1.patdb' in
//         TSDB './tsdb_outdir' with pattern ID 'id1'
//
// Please specify the pattern file with the -pattern_id switch.
```

To select the pattern file with no pattern ID, use the -pattern_id with empty string:

read_patterns -instances [get_instances core_1] -fault_type stuck -pattern_id "" -silent

Example 8

This example demonstrates reporting the auto-loaded pattern files when you do not specify the -silent switch

read_patterns -fault_type stuck

```
// Reading pattern file ./tsdb_outdir/logic_test_cores/
core1.logic_test_core/core1.atpg_mode_internal/core1_internal_stuck.patdb
// Reading pattern file ./tsdb_outdir/logic_test_cores/
core2.logic_test_core/core2.atpg_mode_internal/core2_internal_stuck.patdb
// Reading pattern file ./tsdb_outdir/logic_test_cores/
core3.logic_test_core/core3.atpg_mode_internal/core3_internal_stuck.patdb
// Reading pattern file ./tsdb_outdir/logic_test_cores/
core4.logic_test_core/core4.atpg_mode_internal/core4_internal_stuck.patdb
```

Related Topics

[delete_patterns](#)

[simulate_patterns](#)

[write_patterns](#)

read_procfile

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.

Usage

`read_procfile proc_filename [-Append_or_timing_update | -Replace]`

Description

Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.

For complete information on test procedure file creation, syntax, and structure, refer to “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

Note

 The procfile may contain Tcl variable that will be substituted when the file is read. The referenced Tcl variable must exist in the Tcl global namespace scope. If you are setting the variables within a proc, make sure to use :: as a prefix to the variable name such that it is set in the global namespace. For example, use “set ::my_period 4” such that \$period exits when processing the proc files. For more info about Tcl variable substitution inside the procfile, refer to “[Additional Support for Test Procedure Files](#)” in the *Tessent Shell User’s Manual*.

This command does the following:

- Replaces each existing timeplate with the definition in *proc_filename* for that timeplate.
- Adds any new timeplates defined in *proc_filename*.
- Updates the timing (timeplates) used by existing scan procedures (like `test_setup`, `load_unload` and `shift`) to match the definitions of these procedures in *proc_filename*.
- Merges the named capture procedure (NCP) data in *proc_filename* with NCP data that exists in the tool’s database, as summarized in [Table 5-4](#).

Table 5-4. Named Capture Procedure Merging Under Different Conditions

Condition	-Append_or_timing_update	-Replace
New NCPs ¹ (no NCP in tool has the same name)	Adds new NCPs	Deletes existing NCPs and adds NCPs from <i>proc_filename</i>
NCPs of same name exist in tool but are <i>not</i> used ²	Updates existing NCP’s timing only	Deletes existing NCPs and adds NCPs from <i>proc_filename</i>

Table 5-4. Named Capture Procedure Merging Under Different Conditions

Condition	-Append_or_timing_update	-Replace
NCPs of same name exist in tool and are used ²	Updates existing NCP's timing only	Invalid; results in an error message.
No NCP in <i>proc_filename</i>	Does not alter existing NCPs	Deletes existing NCPs if not used

1. NCP = named capture procedure

2. Used means at least one pattern in the internal or external pattern database uses the NCP.

- Replaces existing clock control definitions of the same name if no test patterns exist. If test patterns exist, an error displays.
- Adds new clock control definitions.

Tip

 A good practice is to regularly use the [write_procfile](#) command to write existing procedures and timing data to a test procedure file when you are experimenting with procedures and timing. This allows you to preserve the most up-to-date procedures and timing data in a form you can use in future runs simply by updating the [add_scan_groups](#) command in your dofile. You can use [read_procfile](#) to recover quickly if existing procedures and timing are altered in an undesirable way.

Arguments

- ***proc_filename***

A required string that specifies the pathname or filename of the test procedure file to read.

- **-Append_or_timing_update**

An optional switch that does the following using the data in *proc_filename*:

- Adds any new NCPs in *proc_filename* to the tool's existing database of NCPs.

The tool treats as new any NCP in *proc_filename* whose name is not identical to that of an NCP in the tool's existing database. An added NCP is available immediately for ATPG use unless you disable it with the "[set_capture_procedures Off](#)" command.

- Update the timing (timeplate) of existing NCPs and other procedures as applicable, without altering them in any other way.

This is the default.

Note

 An NCP in *proc_filename* and its counterpart of the same name in the tool's internal database must have the same event sequence. If their event sequences differ, the command is aborted.

- -Replace

An optional switch that deletes all existing NCPs in the tool’s internal database and replaces them with the NCPs if any in *proc_filename*, provided a current internal or external pattern set does not contain a pattern that uses an existing NCP. If even one existing NCP is used by a pattern, none of them are deleted or replaced. If *proc_filename* does not contain any NCPs, the existing NCPs are deleted and not replaced, provided a current pattern does not use any of them.

When clock control definitions are created automatically by the tool during system mode transition due to execution of the [add_core_instances](#) command to define Tessent OCCs, those clock control definitions are unaffected by issuance of the “*read_procfile proc_filename -replace*” command and switch.

Examples

Example 1

Test procedure file *orig.proc* defines a simple timeplate as well as a shift, load_unload, and named capture procedure *cap1*. The following example uses *orig.proc*, sets the system mode to analysis, then reports the current timeplates and procedures stored in the tool’s database (for brevity, only the parts of the timeplates and procedures relevant to this example are shown):

```

add_scan_groups grp1 orig.proc
set_system_mode analysis
report_timeplate

timeplate gen_tp1 =
    ...
end;

report_procedures

procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
end;

procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
end;

procedure capture cap1 =
    timeplate gen_tp1 ;
    ...
end;

```

Assume a second test procedure file *update.proc* defines a new timeplate in addition to the original one, revises the shift procedure to use the new timeplate, and defines a new named capture procedure *cap2*. The following example adds the new timeplate, updates the timing in the scan procedures, and adds the new named capture procedure. (The timeplate and procedure

changes resulting from the **read_procfile** command are highlighted in bold in the output of the reporting commands.)

read_procfile update.proc

```
// The following occurred at line 46 in file update.proc:  
// Procedure shift updates timing in same procedure from file orig.proc. (W14-1)  
// Loaded new procedure file successfully.  
// 1 new capture procedure is added.  
// 2 total capture procedures are in the system.  
// 2 capture procedures are active for test generation.
```

report_timeplate

```
timeplate gen_tp1 =  
    ...  
end;  
// Timeplate gen_tp1 has an old timeplate  
// timeplate gen_tp1 =  
//    ...  
// end;  
timeplate gen_tp2 =  
    ...  
end;
```

report_procedures

```
procedure shift =  
    scan_group grp1 ;  
    timeplate gen_tp2 ;  
    ...  
end;  
procedure load_unload =  
    scan_group grp1 ;  
    timeplate gen_tp1 ;  
    ...  
end;  
procedure capture cap1 =  
    timeplate gen_tp1 ;  
    ...  
end;  
procedure capture cap2 =  
    timeplate gen_tp2 ;  
    ...  
end;
```

If the **read_procfile** command of the preceding example had included the **-Replace** switch, the original NCP would have been replaced with the new one:

read_procfile update.proc -replace

```
// The following occurred at line 46 in file update.proc:  
// Procedure shift updates timing in same procedure from file orig.proc.  
// (W14)  
// Loaded new procedure file successfully.  
// 1 new capture procedure is added.  
// 1 total capture procedure is in the system.  
// 1 capture procedure is active for test generation.  
  
report_procedures  
  
procedure shift =  
    scan_group grp1 ;  
    timeplate gen_tp2 ;  
    ...  
end;  
procedure load_unload =  
    scan_group grp1 ;  
    timeplate gen_tp1 ;  
    ...  
end;  
procedure capture cap2 =  
    timeplate gen_tp2 ;  
    ...  
end;
```

Related Topics

[report_procedures](#)

[report_timeplate](#)

[write_patterns](#)

[write_procfile](#)

read_sdc

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup, analysis (dft -edt, patterns -scan, patterns -scan_diagnosis and dft -test_points contexts only)

Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.

Usage

```
read_sdc sdc_filename... [{-Instance ins_pathname...} | -Module module_name]
[{:> | >>} file_pathname]
```

Description

Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file. Typically you determine false and multicycle paths as part of static timing analysis before running ATPG.

The read_sdc command parses the SDC file, looking for the set_false_path command, the set_multicycle_path command, and other appropriate commands and their arguments. A summary message displays the following information for the relevant commands found in the SDC file:

- Number of false paths — Paths that cannot be activated in the design's functional (at-speed) mode of operation or are untestable due to test constraints you have applied. False paths are derived from the SDC command set_false_path, which specifies false path timing (the paths that the STA tool should not evaluate for timing).
- Number of multicycle paths — Paths that have a combined time delay through the path greater than the clock period. False paths are derived from the SDC command set_multicycle_path.
- Number of erroneous paths — False paths that are erroneous (for example, due to misnamed pins).
- Number of case analysis — Conditional false paths, which are false paths along with a Boolean condition. If the associated condition is violated, the effect of the false path is considered. Conditional false paths are derived from the Boolean constraints defined by the SDC command set_case_analysis.
- Number of disable timings — Derived from the SDC command set_disable_timing, which disables the specified timing arcs in a circuit and is used to speed up the runtime of the tool. The set_disable_timing command is virtually equivalent to the set_false_path command.
- Number of clock groups — Clock group count.

The SDC commands set_false_path, set_multicycle_path, set_case_analysis, and set_disable_timing are converted into equivalent add_false_paths commands. For example, the following SDC command set_disable_timing:

```
set_disable_timing [get_pins {U1/buf2/Z}]
```

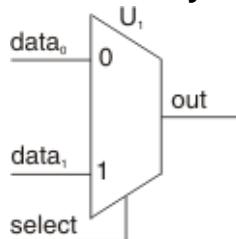
would be translated into the following add_false_paths command:

```
add_false_paths -through U1/buf2/Z
```

The following SDC command set_case_analysis:

```
set_case_analysis 0 [get_pins {U1/select}]
```

Figure 5-3. Case Analysis Example



would be translated into one conditional false path and one unconditional false path:

```
add_false_paths -through U1 if (U1/select != 0)
```

Note

 That is, if U1/select is not 0, the effect of this false path is considered.

```
add_false_paths -through U1/select
```

When test patterns are simulated during ATPG, the related incorrect capture responses are identified and masked (modified to X) in the resultant test patterns, so they are not the source of simulation mismatches when verified in a timing-based simulator.

The read_sdc command supports the following SDC commands:

- create_clock
- create_generated_clock
- set_case_analysis
- set_clock_groups
- set_disable_timing
- set_false_path

- `set_hierarchy_separator`
- `set_multicycle_path`

Be aware that the SDC commands `create_clock` and `create_generated_clock` are completely independent of the command `add_clocks`, which you must still use.

Tip

 To avoid problems extracting correct paths from the SDC specifications, write out the SDC file using the PrimeTime static timing analysis tool. Follow the steps outlined in the section “[Does the SDC File Contain Valid SDC Information?](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information, see “[Pattern Failures Due to Timing Exception Paths](#)” in the *Tessent Scan and ATPG User’s Manual*.

Tip

 You can use the command to list all of the SDC-defined clocks.

You can use the “`set_timing_exceptions_handling -allow_invalid_pin_names on`” command to allow the `read_sdc` command to issue warnings instead of errors when defined objects are not found.

Hybrid TK/LBIST Flow Usage

Reads in the SDC file that describes the false and multicycle paths that should be blocked during logicBIST.

The scan cells that are the destination of false and multicycle paths will be inferred from the SDC file. Re-circulating muxes with inversion will be inserted on their data input pins to ensure these scan cells do not capture values from timing exception paths. Both false paths and multicycle paths are bound in the same way.

Refer to the [Hybrid TK/LBIST Flow User’s Manual](#) for complete information.

Arguments

- **`sdc_filename`**
A required, repeatable string that specifies the pathname of one or more SDC files to read.
- **-Instance `ins_pathname`**
An optional switch and repeatable string that specifies the pathname(s) of one or more design instances to which the information in `sdc_filename` should be applied. Use this switch when the SDC information in the file was developed at the module (block) level and you now want to read it in and have it properly applied to the specified instance(s) of that module in the netlist.

- **-Module *module_name***

An optional switch and string pair that specifies a design module or Tessent Cell library model, to all instances of which the information should be applied in the specified SDC file(s). Use this switch to specify when the SDC information in the file(s) was developed at the module (block) or Tessent Cell library model level and that it should be applied to every instance of that module or model in the netlist.

- **> *file.pathname***

An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file.pathname* with a list of any syntax errors encountered while reading the specified SDC file(s).

- **>> *file.pathname***

An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file.pathname* a list of any syntax errors encountered while reading the specified SDC file(s).

Examples

Example 1

The following example reads the file *new_design_sdc* containing one false path definition and one multicycle path definition in Synopsys Design Constraint (SDC) format:

```
read_sdc new_design_sdc

// Reading SDC file new_design_sdc ...
// Finished reading SDC file new_design_sdc.
// read_sdc summary: 1 false path, 1 multicycle path, 0 erroneous paths
//      0 disable timings, 0 case analysis, 0 clock groups
```

Tip

i If the SDC file contains more than one description of the same path, it is counted only once. Also, the path counts reported in the *read_sdc* summary apply to the current command only; they are not cumulative counts of the total number of false and multicycle path definitions stored in the internal data base of the tool.

Example 2

Continuing with the preceding example, the following example reads another SDC file *latest_design_sdc* containing one false path and three multicycle path definitions, then reports on the false path definitions now stored within the tool:

```
read_sdc latest_design_sdc

// Reading SDC file latest_design_sdc ...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 1 false path, 3 multicycle paths, 0 erroneous paths,
//      0 disable timings, 0 case analysis, 0 clock groups

report_false_paths
```

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2
```

Example 3

The SDC files of the two preceding examples could have been read with one command as in the following example:

```
read_sdc new_design_sdc latest_design_sdc

// Reading SDC file new_design_sdc...
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 2 false paths, 4 multicycle paths, 0 erroneous paths,
//      0 disable timings, 0 case analyses, 0 clock groups
```

Notice the `read_sdc` summary lists the combined totals for all the files read with the one command.

Example 4

Suppose the SDC file `new_design_sdc` of the preceding example contained two false path definitions instead of one, but one of the definitions was erroneous due to misnamed pins. The following example shows sample messaging when this error is detected in the SDC file:

```
read_sdc new_design_sdc latest_design_sdc

// Reading SDC file new_design_sdc ...
// Error: Incorrect pin-instance path name(s) in SDC file new_design_sdc,
//      line 5.
// Error: No gate found for "all_inputs all_outputs"
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 3 false paths, 4 multicycle paths, 1 erroneous path,
//      0 disable timings, 0 case analyses, 0 clock groups

report_false_paths

// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B_input -to_clock all_outputs
// Total reported false paths = 3
```

Notice the tool now has the erroneous path definition stored internally. From here, you might want to check for any additional details about this particular definition:

```
report_false_paths -debug_error -from B_input -to_clock all_outputs

// False Path -from B_input -to_clock all_outputs did not find any
//      gate for "B_input all_outputs"
```

Example 5

The following example deletes the erroneous path definition, enters a corrected version of the definition, then confirms there are no remaining errors in any of the false or multicycle path definitions:

```
delete_false_paths -from B_input -to_clock all_outputs
// Deleted 1 false path

report_false_paths

// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2

add_false_paths -from B -to y[2] y[1] y[0]
report_false_paths

// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B -to y[2] y[1] y[0]
// Total reported false paths = 3

report_false_paths -error

// Total reported false paths = 0
```

Example 6

The following example reads the SDC file *my_block_sdc* and applies the information it contains to instance U1 of the design module *my_block*:

```
read_sdc my_block_sdc -instance U1
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 2 false paths, 0 multicycle paths, 0 erroneous paths
//      0 disable timings, 0 case analyses, 0 clock groups
```

Example 7

The following example reads the file of the preceding example and applies the SDC information to instances U2 and U3 of the same design module:

```
read_sdc my_block_sdc -instance U2 -instance U3
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 4 false paths, 0 multicycle paths, 0 erroneous paths
//      0 disable timings, 0 case analyses, 0 clock groups
```

Example 8

The following example reads the SDC file *my_block_sdc* and applies the information to every instance of the design module *my_block*:

```
delete_false_paths -all
read_sdc my_block_sdc -module my_block

// Reading SDC file my_block_level_sdc ...
// Finished reading SDC file my_block_level_sdc.
// read_sdc summary: 6 false paths, 0 multicycle paths, 0 erroneous paths
//      0 disable timings, 0 case analyses, 0 clock groups
```

Example 9

This example shows that we can read an SDC file in setup mode, transition to analysis mode, and perform X-bounding based on the paths in the SDC file:

```
read_sdc my_design.sdc
set_system_mode analysis
analyze_xbounding
```

Related Topics

[add_false_paths](#)
[report_clocks](#)
[report_false_paths](#)
[report_multicycle_paths](#)
[set_timing_exceptions_handling](#)

read_sdf

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.

Usage

```
read_sdf sdf_filename...
  [-MInimum_delay | -TTypical_delay | -MAximum_delay]
  [-Replace | -Append | -INCremental]
  [-INstance hierarchical_instance_name]
```

Description

Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.

The SDF file is generated by your static timing analysis tool. For complete information, refer to “[Timing-Aware ATPG](#)”, in the *Tessent Scan and ATPG User’s Manual*.

For descriptions of any errors and warnings reported by read_sdf, see “[Errors and Warnings While Reading SDF Files](#)” in the *Tessent Scan and ATPG User’s Manual*.

The following limitations apply to the SDF file:

- For SDF conditional delays, the tool will load them into an internal database. However, only the maximum delay will be used for calculating actual delay. For example:
 - (COND b (IOPATH a y (0.21) (0.54)))
(COND ~b (IOPATH a y (0.27) (0.34)))
 - Timing-aware ATPG uses (0.27) (0.54) regardless of the value of b.
 - Only the first two delay value tokens are used.
- PATHPULSE, PATHPULSEPERCEN, DEVICE, and RETAIN statements are ignored.
- Delays on Memory and Latch are ignored.
- Hold time information is ignored.
- Delays in clock lines are ignored.
- Latch is assumed to be either a launch or capture point, not as a combinational logic in-between.

Arguments

- *sdf_filename*

A required string that specifies the name of the SDF file to read.

- **-MInimum_delay**
An optional switch that loads only the SDF minimum delay data into the tool.
- **-Typical_delay**
An optional switch that loads only the SDF typical delay data into the tool. This is the default.
- **-MAximum_delay**
An optional switch that loads only the SDF maximum delay data into the tool.
- **-REplace**
An optional switch that removes any previously loaded SDF file information before loading the new SDF file information.
- **-APpend**
An optional switch that loads the new SDF file information in addition to the existing information. This is the default. This switch allows you to load multiple SDF files for different modules.

The tool will issue an error message if the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file. Use the **-INCremental** switch if you want to merge all delay data for a flattened pin.

Note

 When using this switch, if the selected delay type (minimum, typical or maximum delay) is different from the previous `read_sdf` command, all previously loaded SDF timing information will be discarded.

- **-INCremental**
An optional switch that loads the new SDF file information in addition to the existing information. If the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file, the new delay data will be cumulated with the previous delay data for that pin. Use the **-APpend** switch if you do not want to merge any delay data for a flattened pin.
- **-INstance *hierarchical_instance_name***
An optional switch and string pair that specifies the hierarchical instance name to which the SDF file is associated. This option is useful when the SDF file is at the module level.

Examples

The following example reads the typical delay data from the `top_worst.sdf` file:

```
read_sdf top_worst.sdf
```

The following example reads the delay data for multiple modules:

```
read_sdf file1.sdf -maximum_delay -instance /top/module1  
read_sdf file2.sdf -maximum_delay -append -instance /top/module2  
read_sdf file3.sdf -maximum_delay -append -instance /top/module3
```

Related Topics

[report_statistics](#)

[set_atpg_timing](#)

read_upf

Context: all contexts

Mode: setup

Loads a UPF (Unified Power Format) file and removes any previously loaded power data.

Usage

`read_upf filename`

Description

Loads a UPF (Unified Power Format) file and removes any previously loaded power data.

You can load multiple UPF files using the native UPF “load_upf” command.

When you use the `read_upf` command, power data is loaded into the tool and the [report_gates](#) command enables power data reporting and shows the power on (PON) or power off (POFF) as well as the power domain of the gate.

The tools support UPF version 2.0.

For information about what happens when you load power data, refer to “[Power-Aware DRC and ATPG](#)” in the *Tessent Scan and ATPG User’s Manual*.

Note

 As described in the [create_dft_specification](#) command description, the DftSpecification is affected by the power domain information to make sure that groups of DesignInstances associated to different power domains are isolated by a SIB. This is done to avoid daisy chaining them with elements from a different power domain and running into issues when accessing the elements in a given power domain while the other power domains are off.

Arguments

- *filename*

A required string that specifies the name of the UPF file.

Examples

Example 1

The following example loads a UPF file:

`read_upf upf_file`

Example 2

In this example, `read_upf` loads power data into the tool and the `report_gates` command reports the power on (PON) status as well as the power domain of the gate.

read_upf *upf_file*

report_gates /lp_case_si_RST_SMS/ati_RST_SYNC/sync_r/U3/Udff

```
//  /lp_case_si_RST_SMS/ati_RST_SYNC/sync_r  sync3msfqxsslul
//  SDI    I(PON)  /vl_sms_lp_case_si_SMS_Proc_SMS_1_stp/U_lp_case_si_SMS_Proc_BIST/uu5/Z
//  D      I(PON)  /lp_case_si_RST_SMS/ati_RST_SYNC/sync_buf/Z
//  SEN    I(PON)  /se
//  CLK    I(PON)  /lp_case_si_RST_SMS/ati_RST_SYNC/uu1/Z
//  Q      O(PON)  /lp_case_si_RST_SMS/ati_RST_SYNC/uu2/A  /lp_case_si_RST_SMS/and_r/B
//  in power domain PD_P2
```

Related Topics

[add_faults](#)

[delete_faults](#)

[report_faults](#)

[read_cpf](#)

[report_power_data](#)

read_verilog

Context: all contexts

Mode: setup

Reads one or more Verilog files into the specified or default logical library.

Usage

dft -no_rtl and patterns -no_rtl Contexts

```
read_verilog [design_path ...] [-verbose] [-force | -no_overwrite_modules]
[-interface_only] [-exclude_from_file_dictionary]
[-no_lib_rescan] [-format {1995 | 2001 | sv31a | sv2005 | sv2009}]
[-pragma_list pragma_list] [-ignore_synthesis_off_sections {On | Off}]
[-ffiles] [-no_duplicate_modules_warnings] [-no_limited_rtl [-report_all_errors]]
```

dft -rtl and patterns -rtl Contexts

```
read_verilog [design_path ...] [-verbose] [-force | -no_overwrite_modules]
[-interface_only] [-exclude_from_file_dictionary]
[-in_library logical_lib_name] [-no_lib_rescan] [-no_mfcu]
[-format {1995 | 2001 | sv31a | sv2005 | sv2009}]
[-pragma_list pragma_list] [-ignore_synthesis_off_sections {On | Off}]
[-vcs_compatibility] [-ffiles] [-no_duplicate_modules_warnings]
```

Description

Reads one or more Verilog files into the specified or default logical library.

You define the list of available libraries with the [set_logical_design_libraries](#) command which you only use in rtl context; only one library exists in no_rtl context and it is named “work”.

Note that the MGC_TESSENT and SYNTHESIS design macros are defined by default. For more information, refer to the [set_design_macros](#) command description.

When in -no_rtl context ([set_context context -no_rtl](#)), the tool issues a warning when parsing RTL constructs. See the -no_limited_rtl switch for additional guidance.

Arguments

- *design_path*

One or more optional files containing Verilog elements to load. A well formatted Tcl list of filenames can also be supplied, such as the list returned by the Tcl glob command. The names of the files can contain wildcards. The *design_path* is optional if you use the -f switch to specify a file that contains at least one *design_path*.

If you have used the [set_design_sources](#) command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- **-verbose**

An optional switch that reports each file read. For every file read, including the ‘include’ files, the following messages are returned:

```
// Reading Verilog file filename  
// Finished reading file filename
```

- **-force**

An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using `set_current_design` and you add any configuration information to it (for example, `add_clocks` or `add_scan_chains`), executing the `read_verilog` command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you execute the command with the `-force` switch to indicate that it is acceptable to delete the setup data.

- **-no_overwrite_modules**

An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- **-interface_only**

An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters.

- **-exclude_from_file_dictionary**

An optional switch used when you input a Verilog simulation model into Tessent Shell, and the model does not have a `celldefine; issuing this switch ensures that the read is excluded in the file dictionary and, consequently, excluded in the DC shell load script.

- **-in_library *logical_lib_name*** (“`set_context context -rtl`” only)

An optional switch that specifies which logical library the modules are to be read into. The list of possible logical libraries, including the default library, is defined by the `set_logical_design_libraries` command. If this option is not specified, the modules are read into the default library.

- **-no_lib_rescan**

An optional switch that specifies to not rescan the `-y` directories and the `-v` files from the beginning each time, but to search for the module beginning with the directory or file in which the previous module was found.

- **-no_mfcu** (“`set_context context -rtl`” only)

An optional switch that applies to SystemVerilog formats which specifies to compile each file into a separate compile unit. By default, the tool compiles into a multiple-file unit.

- **-format {1995 | 2001 | sv31a | sv2005 | sv2009}**

An optional switch that specifies the format of the input Verilog files. The default is 2001. In a mixed-format situation, you should be sure to re-define your -y and -v library search path when loading modules in a new format because you cannot mix formats within a single `read_verilog` command invocation.

In the `-no_rtl` context, you must use this switch together with the `-interface_only` switch to read modules that have interfaces without special System Verilog ports.

In the `-rtl` context, you can use the `-format` switch with or without the `-interface_only` switch.

The `sv31a` is not a standard format but it is the default inside Design Compiler when `hdlin_sv_packages` is not defined.

- **-pragma_list *pragma_list***

An optional switch and string pair that defines the pragmas for the input files. The `pragma_list` is a Tcl list containing one or more objects. When this option is not specified, the following list of strings is used as default pragmas: {`pragma synthesis mgc_tessent synopsys LV_pragma mentor`}. You can override the default pragmas by specifying a new pragma list as shown here:

```
read_verilog 1.v -pragma_list {my_pragma1 mgc_tessent}
```

You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

```
read_verilog 1.v -pragma_list {}
```

Pragmas are special comments in the source. They control how the tool interprets the source. This example of source code uses a pragma pair to turn translation on or off:

```
module ram1 ( ...
  ...
  // synopsys translate_off
  myram u_atpg ( ... );
  // synopsys translate_on
```

By default, `synopsys` is a pragma. The tool will ignore all input from when it reads the `pragma` directive “`// synopsys translate_off`” until the “`// synopsys translate_on`” `pragma` directive. The tool does not include the `myram` instance.

If you use the command, “`read_verilog` with `-pragma_list {}`”, you unset the default pragmas and the tool includes the `myram` instance.

- **-ignore_synthesis_off_sections {On|Off}**

An optional switch and literal pair that specifies whether to ignore the `synthesis_off` sections in the input files. The default is On.

On — A literal that specifies to ignore `synthesis_off` sections and to NOT compile the code found within the `synthesis_off/on` block.

Off — A literal that specifies to not ignore the `synthesis_off` sections and to compile the code found within the `synthesis_off/on` block.

- **-vcs_compatibility** (“*set_context context -rtl*” only)

An optional switch that specifies to ignore a series of syntax errors allowed by the Synopsys compilers. By default, the tool reports these non-compliant usages as errors.

When specified, this setting is in effect until you execute the *delete_design*. You only specify this switch in the first execution of the *read_verilog* command after tool invocation, or after the *delete_design* command is executed.

- **-f files**

An optional switch and Tcl list that specifies the name(s) of one or more ASCII files that contains pointers to design files. The referenced file is usually generated by another tool, so the *-f* option allows you to conveniently use the file rather than to reformat the contents of the file into a *read_verilog* command line. The file can contain only the following elements, one per line:

- *design_path* — A string that specifies the name of a Verilog file to load.
- *-v design_path* — A string that specifies the name of a Verilog file to load.
- *-y dir_path* — A string that specifies a path to a directory that contains Verilog files to load.
- *+libext+file_extension* — A string that specifies which files in the “*-y dir_path*” to load. For example: “*+libext+.v*” loads all *.v* files.
- *+define+XXX[=YYY]* — A string that defines a macro called “*XXX*” with an optional value of “*YYY*”. The defined macro is used when compiling any files listed in the *-f* file.
- *+incdir+dir_path1+dir_path2+...* — A string that defines one or more directory paths to directories that contain files that use the ‘*include*’ directive.

- **-no_duplicate_modules_warnings**

An optional switch that suppresses overwrite warnings when loading a netlist.

- **-no_limited_rtl [-report_all_errors]** (“*set_context context -no_rtl*” only)

An optional switch to flag an error when parsing RTL when the tool is in *-no_rtl* context. When not specified, RTL Verilog constructs are flagged with a warning.

The environment variable TESSENT_NO_LIMITED_RTL can be used instead of the *-no_limited_rtl* switch. Verilog attributes are ignored with a warning by default and with the switch. The optional *-report_all_errors* switch reports all of the RTL construction errors found by the tool in all modules. Usage is as follows:

- *-no_limited_rtl* — When the tool encounters the RTL construct, the tool reports one error per module for all files.
- *-no_limited_rtl -report_all_error* — The tool reports all the RTL construct errors that are encountered in all the modules in all files.

- TESSENT_NO_LIMITED_RTL = 1 (it can be any value except "report_all_errors")
 - When the tool encounters the RTL construct, the tool reports one error per module for all modules in all files.
- TESSENT_NO_LIMITED_RTL = report_all_errors (it can be upper case or lower case) — The tool reports all the RTL construct errors which are encountered in all the modules in all files.

Examples

Example 1

The following example loads all Verilog files matching *.v inside the directory src:

```
read_verilog src/*.v
```

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the edited_design directory.

```
SETUP> set_context dft -rtl
# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work will be mapped to a scratch dir

SETUP> set_logical_design_libraries -logical_library_list { library1 work}
# Define macro1 for Verilog compilation

SETUP> set_design_macros macro1=5
# Search for Verilog modules in /design/abc/*.v

SETUP> set_design_sources -y /design/abc -extension v
# Read Verilog files

SETUP> read_verilog top.v core.v -in_library work -format 2001
# Read VHDL files

SETUP> read_vhdl core1.vhdl -in_library work -format 1993
SETUP> read_vhdl {core2.vhdl core3.vhdl} -in_library library1 -format 1993
# Set the top

SETUP> set_current_design top
# Enter insertion mode

SETUP> set_system_mode insertion
# Perform design query and manipulation commands

INSERTION> ...
# Write the design
```

INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit

Example 3

The following example uses the *-f files* switch to load in the design file(s) specified within an ASCII file that contains spaces in the filename:

SETUP> read_verilog -f [list “updated cores.f”]

Example 4

The following example uses the *-f files* switch to load in the design files specified within two different ASCII files:

SETUP> read_verilog -f {block_a_cores.f block_b_cores.f}

Related Topics

[delete_design](#)

[read_cell_library](#)

[read_flat_model](#)

[set_current_design](#)

[read_vhdl](#)

[set_design_macros](#)

[set_design_sources](#)

[set_design_include_directories](#)

[set_logical_design_libraries](#)

read_vhdl

Context: all contexts

Mode: setup

Reads one or more VHDL files into the specified or default logical library.

Usage

dft -no_rtl and patterns -no_rtl Contexts

```
read_vhdl [design_path] [-interface_only] [-exclude_from_file_dictionary]  
[-force | -no_overwrite_modules] [-format {1987 | 1993 | 2002 | 2008}]  
[-pragma_list pragma_list] [-ignore_synthesis_off_sections {on | off}]  
[-ffiles] [-no_duplicate_modules_warnings]
```

dft -rtl and patterns -rtl Contexts

```
read_vhdl [design_path] [-interface_only] [-exclude_from_file_dictionary]  
[-force | -no_overwrite_modules] [-in_library logical_lib_name]  
[-format {1987 | 1993 | 2002 | 2008}] [-pragma_list pragma_list]  
[-ignore_synthesis_off_sections {on | off}] [-ffiles]  
[-no_duplicate_modules_warnings]
```

Description

Reads one or more VHDL files into the specified or default logical library. The list of possible logical libraries, including the default library, is defined by the [set_logical_design_libraries](#) command.

You can execute this command with the -force switch but without a *design_path* argument to clear design setup data only and not read any new files.

Arguments

- *design_path*

An optional Tcl list that specifies the name(s) of one or more VHDL files to load. The names of the files can contain wildcards. The *design_path* option is optional if you use the -f switch to specify a file that contains at least one *design_path*.

This command includes an example that shows a suggested way of loading different VHDL files into different libraries.

- -interface_only

An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters.

- **-exclude_file_dictionary**

An optional switch that is used when you input a VHDL simulation model into Tessent Shell, and the model does not have a `celldefine; issuing this switch ensures that the read is excluded from the file dictionary and, consequently, excluded in the DC shell load script.

- **-force**

An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), executing the read_vhdl command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you execute the command with the -force switch to indicate that it is acceptable to delete the setup data.

- **-no_overwrite_modules**

An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- **-in_library *logical_lib_name*** (“set_context context -rtl” only)

An optional switch that specifies which logical library the modules are to be read into. The list of possible logical libraries, including the default library, is defined by the set_logical_design_libraries command. If this option is not specified, the modules are read into the default library. This switch is only available in -rtl context.

- **-format {1987 | 1993 | 2002 | 2008}**

An optional switch that specifies the expected format of the input files and which VHDL language constructs can be understood during parsing. The default is 1993.

- **-pragma_list *pragma_list***

An optional switch and string pair that defines the pragmas for the input files. The pragma_list is a Tcl list containing one or more objects. When this option is not specified, the following list of strings is used as default pragmas: {pragma synthesis mgc_tessent synopsys LV_pragma mentor}. You can overwrite the default pragmas by specifying a new pragma list as shown here:

```
read_vhdl 1.v -pragma_list {my_pragma1 mgc_tessent}
```

You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

```
read_vhdl 1.v -pragma_list {}
```

Pragmas are special comments in the source. They are used to control how the tool interprets the source. This example of source code uses a pragma pair to turn translation on and off:

```
begin
  -- synthesis translate_off
  myram: work.comp_decls.topmem port map(...);
  -- synthesis translate_on
```

If synthesis is specified as a pragma, either by using the `-pragma_list {synthesis}` switch pair for `read_vhdl` or because it is a default pragma, the tool will recognize the “`-- synthesis translate_off`” directive and will ignore all input from the source until it encounters the “`-- pragma translate_on`” directive. The result will be, in this case, that `myram` will not be included.

If you issue a `read_vhdl` with `-pragma_list {}`, the default pragmas will be unset and the instance `myram` will be included by the tool.

- `-ignore_synthesis_off_sections {On | Off}`

An optional switch and literal pair that specifies whether to ignore the `synthesis_off` sections in the input files. The default is On.

On — A literal that specifies to ignore `synthesis_off` sections and to NOT compile the code found within the `synthesis_off/on` block.

Off — A literal that specifies to not ignore the `synthesis_off` sections and to compile the code found within the `synthesis_off/on` block.

- `-f files`

An optional switch and Tcl list that specifies the name(s) of one or more ASCII files that contains pointers to design files to load. The names of the files can contain wildcards. The referenced file is usually generated by another tool, so the `-f` option allows you to conveniently use the file rather than to reformat the contents of the file into a `read_vhdl` command line. The file can contain only the following elements, one per line:

- `design_path` — A string that specifies the name of a VHDL file to load.
- `-no_duplicate_modules_warnings`

An optional switch that suppresses overwrite warnings when loading a netlist.

Examples

Example 1

The following example instructs the tool to read only the interface of the modules/entities and ignore the contents:

```
set_context dft -no_rtl
read_verilog data/pll.v -interface_only
read_vhdl {data/ram_bist.vhd data/edt_ip*.vhf} -interface_only
```

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the *edited_design* directory.

```
SETUP> set_context dft -rtl
# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work will be mapped to a scratch dir

SETUP> set_logical_design_libraries -logical_library_list { library1 work}

# Define macro1 for Verilog compilation

SETUP> set_design_macros macro1=5

# Search for Verilog modules in /design/abc/*.v

SETUP> set_design_sources -y /design/abc -extension v

# Read Verilog files

SETUP> read_verilog top.v core.v -in_library work -format 2001

# Read VHDL files

SETUP> read_vhdl core1.vhdl -in_library work -format 1993
SETUP> read_vhdl {core2.vhdl core3.vhdl} -in_library library1 -format 93

# Set the top

SETUP> set_current_design top

# Enter insertion mode

SETUP> set_system_mode insertion

# Perform design query and manipulation commands

INSERTION> ...
# Write the design

INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit
```

Example 3

The following example uses the *-f files* switch to load in the design file(s) specified within an ASCII file that contains spaces in the filename:

```
SETUP> read_vhdl -f {updated cores.f}
```

Example 4

The following example uses the *-f files* switch to load in the design files specified within two different ASCII files:

```
SETUP> read_vhdl -f {block_a_cores.f block_b_cores.f}
```

Related Topics

[delete_design](#)
[read_icl](#)
[read_verilog](#)
[set_current_design](#)

read_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Reads a DFTVisualizer preferences file and sets current preferences as described in the file.

Usage

`read_visualizer_preferences filename`

Description

Reads a DFTVisualizer preferences file and sets current preferences as described in the file.

The `read_visualizer_preferences` command reads a DFTVisualizer preference file previously saved with the [write_visualizer_preferences](#) command and updates the current DFTVisualizer session to use the preferences in the specified file.

Tip

 When DFTVisualizer first opens in a tool session, it reads the `$HOME/.DftVisualizerrc` preference file if it is present and sets the preferences accordingly; otherwise, it uses hard-coded defaults. Current preference settings are written to this file when you click the Write Prefs button at the bottom of the DFTVisualizer Preferences window.) DFTVisualizer uses the preferences in the default file until overridden by a `read_visualizer_preferences` command. The last such command entered establishes the preferences DFTVisualizer uses.

Arguments

- *filename*

A required string that specifies the name of the preference file to read.

Examples

The following example reads the DFTVisualizer preference file, `my_prefs.dftvis`, and updates the current tool session to use the preferences specified in the file:

```
set_system_mode analysis  
read_visualizer_preferences my_prefs.dftvis
```

Related Topics

[write_visualizer_preferences](#)

read_window_contents

Context: all contexts

Mode: setup, analysis

Reads the contents of the specified file into a DFTVisualizer window.

Usage

```
read_window_contents file.pathname -display window_name [-format format]
```

Description

Reads the contents of the specified file into a DFTVisualizer window. The file specified by the *file.pathname* argument must be in a supported format as described in the *-format* argument description.

The **-display** argument specifies which DFTVisualizer window is opened (if needed) and into which the *file.pathname* content is to be read.

Arguments

- ***file.pathname***

A required string that specifies the pathname of the file that is to be read into a DFTVisualizer window. If the file is to be read into a Flat Schematic or Hierarchical Schematic window, the file must be in the Mentor Graphics Tessent schematic format as generated by the [write_window_contents](#) command.

- **-display *window_name***

A required switch and literal pair that specifies the DFTVisualizer window into which the *file.pathname* content is to be read. Select the window name from the following list:

FLAt_schematic — A literal that specifies the Flat Schematic window. This is the default.

HIErarchical_schematic — A literal that specifies the Hierarchical Schematic window.

Text_editor — A literal that specifies the Text Editor window.

- **-format *format***

An optional switch and literal pair that specify the format of the file to be read. The file displays in the Text Editor window with appropriate highlighting and formatting. This option is not valid with **-display FLAt_schematic** or **HIErarchical_schematic** arguments which require Mentor Graphics Tessent schematic file format. Switch options for *-format* include:

TEXt — A literal that specifies an ASCII text format. This is the default.

Verilog — A literal that specifies the Verilog format.

VHdl — A literal that specifies the VHDL format.

TESt_procedure — A literal that specifies the Test Procedure format.

Dofile — A literal that specifies the Dofile format.

Related Topics

[close_visualizer](#)

[open_visualizer](#)

[set_visualizer_preferences](#)

[write_window_contents](#)

register_attribute

Context: unspecified, all contexts

Mode: all modes

Registers a new user-defined attribute.

Usage

```
register_attribute -name attribute_name -value_type data_type -default default_value
    [-min min_value] [-max max_value] [-enum enum_values] -object_types type_list
    [-dense] [-description description_text] [-show_default] [-hidden]
```

Description

Registers a new user-defined attribute.

This command allows you to define new attributes for object types available in Tesson Shell. After registration, the value of the new attribute can be set and retrieved on objects such as a module or an instance.

If an attribute with the same name already exists for the specified object type, or if an object type in *type_list* does not exist, or if a data type is invalid for one of the options, an error message displays.

If you want an attribute set on a module to be visible (inherited) on the instances of the module, register the attribute only on the *module* object type and not on the instance object type. The same is true for inheritance between ports and pins and between nets and pins. The following exceptions to this inheritance rule exist for Boolean attributes:

- For Boolean attributes registered on module and instance object types — When the value is queried on an instance and it is the default value (usually false), the value from the corresponding module is returned.
- For Boolean attributes registered on port and pin object types — When the value is queried on a pin and it is the default value (usually false), the value from the corresponding port is returned.

Arguments

- **-name *attribute_name***

A required switch and value pair that specifies the name of the attribute to register. An attribute name must begin with an alpha character and must only contain alpha-numeric characters and the underscore (_).

- **-value_type *data_type***

A required switch and value pair that specifies the attribute's data type. The -value_type argument is an enumerated list with the following possible values: Boolean, integer, double, and string.

- **-default *default_value***

A required switch and value pair that specifies the default value for the attribute when the attribute is not explicitly set. This pair is required for attributes with non-Boolean data types only; it cannot be set for attributes whose data type is Boolean. Attributes with a data type of Boolean automatically default to false.

- **-min *min_value***

An optional switch and value pair that specifies the minimum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is greater than or equal to *min_value*. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see “[Attribute Value Types](#).”

- **-max *max_value***

An optional switch and value pair that specifies the maximum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is less than or equal to *max_value*. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see “[Attribute Value Types](#).”

- **-enum *enum_values***

An optional switch and value pair that specifies a Tcl list of possible values for the attribute. The *enum_values* value can only be set to one value in the list. This argument is only valid for attributes whose data type is string.

- **-object_types *type_list***

A required switch and value pair that specifies the object type to which the new attribute can be assigned. The *type_list* string is a Tcl list that contains one or more of the following object types: module, instance, port, pin, net, or icl_module. If an object type is not included in the *type_list*, the new attribute cannot be assigned to that object_type.

Extra data types will be available in subsequent releases and the *type_list* will be augmented as they become available.

- **-dense**

An optional Boolean switch that specifies the storage type for the attribute.

When not specified — Specifies *sparse* storage which only stores values for objects that have an attribute value set to something different than the default value. This option is a good choice when many of the attributes on an object are set to their default value because it saves memory. However, indexing and searching can be slower and end up consuming more memory if the number of objects having the attribute specified is more than 10% of the object count.

When specified — Specifies *dense* storage which creates an entry for each object and its attribute value. This option is a good choice when many objects with the same attribute have a distributed set of attribute values because it enables faster indexing and searching. However, it has a higher memory usage when less than 10% of the objects have the attribute specified.

- **-description** *description_text*

An optional switch and value pair that specifies to store a string describing the attribute. The description is displayed when the [report_attributes](#) command is executed for the attribute.

The *description_text* argument is a quoted string that can be formatted as a Tcl string which includes support for escaped characters such as “\n” and “\t”.

- **-show_default**

An optional Boolean switch that specifies how the registered attribute is handled by the [get_attribute_list](#) and [report_attributes](#) commands.

false — If set to false during registration, the “[get_attribute_list](#) *obj_spec*” and “[report_attribute](#) *obj_spec*” command executions list or report attributes found on any object within *obj_spec* only if the attribute is set to a non default value.

true — If set to true, the attribute is always listed or reported if *obj_spec* contains at least one object of the type for which the attribute is registered for.

- **-hidden**

An optional switch that specifies the attribute should not be listed or reported by the [get_attribute_list](#) and [report_attributes](#) commands. The attribute can be set and queried using the [set_attribute_value](#) and [get_attribute_value_list](#) commands. You use this switch when you want to keep an attribute hidden. For example, while you develop a new feature and want to only make the attribute visible when you are ready to release the new feature.

Examples

Example 1

The following example registers the attribute named speed with a range of possible values equal to or greater than 100 and less than or equal to 500; assigns a default value of 200 to it; sets it as an attribute for the pin design object type only. Notice how the Tcl append command is used to allow typing the string on multiple lines without inserting extra white spaces into the string.

```
set desc "This attribute is used to hold the target frequency the pin is \n"
append desc "allowed to have.\n"
append desc "The allowed frequency range of 100 to 500 was chosen to match\n"
append desc "the technology spec."
register_attribute -name speed \
    -value_type integer \
    -default 200 \
    -min 100 \
    -max 500 \
    -object_types pin \
    -description$desc
```

Example 2

The following example defines the attribute named added_by as a string with a value of either “John”, “Patrick”, or “Susan” with a default of John, and a design object type of module:

```
register_attribute -name added_by \
    -value_type string \
    -default John \
    -enum {John Patrick Susan} \
    -object_types net
```

Related Topics

[get_attribute_list](#)
[get_attribute_option](#)
[get_attribute_value_list](#)
[reset_attribute_value](#)
[report_attributes](#)
[set_attribute_options](#)
[set_attribute_value](#)
[unregister_attribute](#)

register_callback

Context: unspecified, all contexts

Mode: all modes

Register a Tcl proc which is to automatically run before or after a built-in feature of Tesson Shell.

Usage

```
register_callback feature_identifier proc_name
    [-order integer] [-static_arguments list_of_argument_values]
```

Description

The register_callback command registers a Tcl proc to automatically run before or after a Tesson Shell builtin feature runs. You use this command in a Tcl file located at *tcl_modules/your_company_name/any_name.tcl* file in one of your [plugin directory](#).

Once you have created and registered a callback procedure to run before or after a specific feature, the proc behaves as if it is part of the builtin behavior of the tool.

You can display the run time for the callbacks you have registered by setting the following environment variable prior to invoking the tool:

```
TESSENT_DISPLAY_CALLBACKS_RUN_TIME 1
```

When set to '1', the tool displays the execution time of those callbacks registered by you.

Arguments

- ***feature_identifier***

A special string that identifies the Tesson Shell builtin feature before or after which you want your proc to run. The following lists all supported feature names and the arguments your proc must have:

- `create_dft_specification.post`

Registers a Tcl proc that automatically runs after the DftSpecification wrapper has been created within the [create_dft_specification](#) command.

If a proc called “`create_dft_specification.post`” is declared in the global namespace, the proc is automatically run after all other procs registered for that feature. If you explicitly register the proc called “`create_dft_specification.post`” in the global namespace, the proc runs at the specified order value instead of at the end.

The proc must be declared with these arguments: `{root_wrapper args}`.

The `root_wrapper` argument holds the value of the DftSpecification wrapper that was created. The wrapper can then be modified using the configuration editing commands listed in [Table 10-1](#) on page 3175.

This callback is useful to automatically apply custom edits to the created DftSpecification wrapper.

- `create_flat_model.post`

Registers a Tcl proc that automatically runs after the flat model is created. The proc must be declared with this argument: `{args}`.

This callback is useful to attach attributes on the newly generated gate_pin objects.

The “`get_system_mode -is_in_transition`” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on `-is_in_transition`, see the [get_system_mode](#) command description.

- `create_flat_model.pre`

Registers a Tcl proc that automatically runs before the flat model is created. The proc must be declared with this argument: `{args}`.

This callback is useful to insert checks that you want to enforce prior to generating the flat model. You can also use it to set the `preserve_boundary` attribute (see “[Built-In Attributes](#)” on page 3058) on hierarchical pins you want preserved in the flat_model. To block the creation of the flat model, issue the error condition with the `display_message` command and terminate the proc with the “`return -code error`” command.

The “`get_system_mode -is_in_transition`” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on `-is_in_transition`, see the [get_system_mode](#) command description.

- `create_patterns_specification.post`

Registers a Tcl proc that automatically runs after the [PatternsSpecification](#) wrapper has been created within the `create_patterns_specification` command.

If a proc called “`create_patterns_dft_specification.post`” is declared in the global namespace, it is automatically run after all other procs registered for that feature. If you explicitly register the proc called “`create_patterns_specification.post`” in the global namespace, it will run at the specified order value instead of at the end.

The proc must be declared with these arguments: `{root_wrapper args}`.

The `root_wrapper` argument holds the value of the [PatternsSpecification](#) wrapper that was created. The wrapper can then be modified using the configuration editing commands listed in [Table 10-1](#) on page 3175.

This callback is useful to automatically apply custom edits to the created [PatternsSpecification](#) wrapper.

- `delete_design.pre`

Registers a Tcl proc that automatically runs before the design is deleted. The proc must be declared with these arguments: {args}.

This callback is useful to insert checks that you want to enforce prior to deleting the design data. To block the deletion of the design, issue the error condition with the `display_message` command and terminate the proc with the “`return -code error`” command.

- `elaboration_creation.post`

Registers a Tcl proc that automatically runs after the design is elaborated. The proc must be declared with this argument: {args}.

This callback is useful to automatically perform actions on the newly-elaborated design such as setting attributes on the design objects. The `get_current_design` command can be used within this proc to get the module that was elaborated.

The “`get_system_mode -is_in_transition`” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on `-is_in_transition`, see the [get_system_mode](#) command description.

- `elaboration_creation.pre`

Registers a Tcl proc that automatically runs before the design is elaborated. The proc must be declared with this argument: {args}.

The args consist of a dictionary containing two keys: `design_name` and `library_name`. The `design_name` key holds the name of the design that is going to be elaborated. The `library_name` contains the name of the logical library in which the design being elaborated exists.

This callback is useful to automatically perform actions before the design is elaborated such as reading some extra design files.

The “`get_system_mode -is_in_transition`” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on `-is_in_transition`, see the [get_system_mode](#) command description.

- `exit.pre`

Registers a Tcl proc that automatically runs before the tool exits.

The proc must be declared with this argument: {args}.

Use “`array set ARGS $args`” to set the arguments into an array and “`$ARGS(force)`” to know if the `-force` Boolean option was specified or not. The `ARGS(force)` value is 1 when the `-force` Boolean switch of the `exit` command was specified and 0 when it was not.

This callback is useful to automatically perform checks that you want to enforce prior to exiting the tool. To block the exit of the tool, issue the error condition with the [display_message](#) command and terminate the proc with the “return -code error” command.

- **extract_icl.post**

Registers a Tcl proc that automatically runs after ICL extraction has completed. This includes the writing of the ICL to file.

The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

This callback allows you to post process any files written out during ICL extraction. If you need to add attributes to ICL objects before the ICL is written out to file, use the extract_icl.pre_write callback.

- **extract_icl.pre**

Registers a Tcl proc that automatically runs before any check and setup of the design to ensure proper ICL extraction.

The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

This callback allows you to perform your own check and setup of the design to ensure successful ICL extraction and assuming that you have added some DFT circuitry yourself.

- **extract_icl.pre_write**

Registers a Tcl proc that automatically runs after ICL extraction has completed but before the writing of the ICL to a file. The callback is called even if [extract_icl -write_in_tsdb](#) is set to off.

The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

This callback is useful to add attributes to ICL objects before the ICL is written out to file. To perform actions on the written out file, use the extract_icl.post callback that is called at the very end of the ICL extraction feature.

- **insert_test_logic.post**

Registers a Tcl proc that automatically runs after the [insert_test_logic](#) has done all its actions including saving the design.

The proc must be declared with this argument: {args}.

This callback is useful to do design introspection and generate custom reports.

- **insert_test_logic.post_insertion**

Registers a Tcl proc that automatically runs after the [insert_test_logic](#) has done all its insertion actions but before it saves the design.

The proc must be declared with this argument: {args}.

This callback is useful to do custom editing to the netlist which you want to be part of the saved netlist.

- o [insert_test_logic.pre](#)

Registers a Tcl proc that automatically runs after the [insert_test_logic](#) has done all its insertion actions but before it saves the design.

The proc must be declared with this argument: {args}.

This callback is useful to do custom checks prior to letting the [insert_test_logic](#) do its job. If your check fails, then you issue error messages with the “[display_message -error](#)” command and return with -code error to cause the [insert_test_logic](#) operations to be interrupted. The user will be left in analysis mode, able to fix the issues reported by your custom checks.

- o [module_matching.post](#)

Registers a Tcl proc that automatically runs after the module matching feature was performed. See the [set_module_matching_options](#) and the [set_design_sources](#) command description for more information about this feature.

The proc must be declared with this argument: {args}.

This callback is useful to set attribute on the design objects from the matched ICL or TCD views—see “[Tessent Core Description](#)” on page 3755. For example, you can use the `get_ijtag_instances` or “`get_modules -filter instrument_name`” commands to get the list of design instances or modules for which there is a matched ICL module then use the ICL introspection commands to transfer information from ICL to the associated design objects.

The “`get_system_mode -is_in_transition`” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on `-is_in_transition`, see the [get_system_mode](#) command description.

- o [module_matching.pre](#)

Registers a Tcl proc that automatically runs before the module matching feature is performed. See the [set_module_matching_options](#) and the [set_design_sources](#) command description for more information about this feature.

The proc must be declared with this argument: {args}.

This callback is useful to adjust the `set_design_sources` and/or the `set_module_matching_option` settings prior to doing the module matching.

The “get_system_mode -is_in_transition” command will return a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the [get_system_mode](#) command description.

- [process_dft_specification.post_insertion](#)

Registers a Tcl proc that automatically runs after the insertion of the instruments was performed within the [process_dft_specification](#) command.

If a proc called “process_dft_specification.post_insertion” is declared in the global namespace, it is automatically run after all other procs registered for that feature. If you explicitly register the proc called “process_dft_specification.post_insertion” in the global namespace, it runs at the specified order value instead of at the end.

The proc must be declared with those arguments: {root_wrapper args}.

The root_wrapper argument holds the value of the DftSpecification wrapper that was processed.

This callback is useful to automatically apply custom edits to the design after all instruments were inserted but prior to when the modified design is written out into the dft_inserted_design directory.

- [process_patterns_specification.post_open](#)

Registers a Tcl proc that automatically runs immediately after the [open_pattern_set](#) command associated to every TestStep or ProcedureStep wrapper found in the processed PatternsSpecification.

The proc must be declared with those arguments: {config_path args}. The config_path argument holds the path to the TestStep or ProcedureStep wrapper being processed.

This callback is useful to add debug code that introspects the initial state of the network prior to running the iCalls associated to the given TestStep or ProcedureStep wrapper. Commands such as [get_iclock_list](#), [get_iclock_option](#), and [get_pattern_set_option](#) are used to introspect the relevant information to echo to the screen.

- [process_patterns_specification.pre_close](#)

Registers a Tcl proc that automatically runs immediately before the [close_pattern_set](#) command associated to every [TestStep](#) or [ProcedureStep](#) wrapper found in the processed [PatternsSpecification](#).

The proc must be declared with these arguments: {config_path args}. The config_path argument holds the path to the TestStep or ProcedureStep wrapper being processed.

This callback is useful to add debug code that introspects the end state of the network after having run the iCalls associated to the given TestStep or ProcedureStep wrapper. Commands such as [get_iclock_list](#), [get_iclock_option](#), and [get_pattern_set_option](#) are used to introspect the relevant information to echo to the screen.

- [set_context.post](#)

Registers a Tcl proc that automatically runs after the context was changed.

The proc must be declared with these arguments: {context args}. The context argument hold the name of the context that was just entered.

This callback is useful to automatically set the context upon entering that context.

- [set_context.pre](#)

Registers a Tcl proc that automatically runs before the context is changed.

The proc must be declared with these arguments: {context args}. The context argument holds the name of the context that is to be entered.

This callback is useful to automatically perform checks that you want to enforce prior to changing context. To block the context change, issue the error condition with the `display_message` command and terminate the proc with the “return -code error” command.

- [set_current_mode.post](#)

Registers a Tcl proc that automatically runs after the current mode was changed.

The proc must be declared with these arguments: {mode_name args}. The mode_name argument hold the name of the mode that was just set.

This callback is useful to automatically perform automatic setting when the current mode was changed. See the [set_current_mode](#) command for more information about the current mode setting.

- [set_current_mode.pre](#)

Registers a Tcl proc that automatically runs before the current mode is changed.

The proc must be declared with these arguments: {mode_name args}. The mode_name argument hold the name of the mode that is to be set.

This callback is useful to automatically perform checks that you want to enforce prior to changing the current the mode name. To block the mode name change, issue the error condition with the `display_message` command and terminate the proc with the “return -code error” command.

- [system_mode_transition.post](#)

Registers a Tcl proc that automatically runs after the system mode was changed.

The proc must be declared with these arguments: {from to args}. The “from” argument hold the name of the system mode prior to the system mode transition while the “to” argument holds the name of the system mode after the system mode transition.

This callback is useful to automatically perform automatic setting automatically after the system mode was changed. Use the get_context command to configure your actions to specific contexts.

- system_mode_transition.pre

Registers a Tcl proc that automatically runs before the system mode is changed.

The proc must be declared with these arguments: {from to args}. The “from” argument hold the name of the system mode prior to the system mode transition while the “to” argument holds the name of the system mode after the system mode transition.

This callback is useful to automatically perform checks that you want to enforce prior to changing the system mode. To block the system mode transition, issue the error condition with the display_message command and terminate the proc with the “return -code error” command. Use the get_context command to configure your actions to specific contexts.

- write_edt_files.post_insertion

Registers a Tcl proc that automatically runs after the insertion of the EDT instruments was performed within the [write_edt_files](#) command.

The proc does not take any arguments.

This callback is useful to automatically apply custom edits to the design after EDT instruments were inserted but prior to when the modified design is written out into the [dft_inserted_designs](#) directory.

- **proc_name**

Specifies the name of the proc to call when the specified feature is happening. The proc name can be with a fully specified namespace prefix or not. When the proc name does not start with ::, it is search relative to the current namespace. Using “[namespace current]::proc_name” is equivalent to specifying proc_name.

- **-order integer**

Specifies the execution order relative to the other callbacks registered for the same feature. When unspecified, the order defaults to 5. You can specify any integer between 1 and 10. A callback with an order value of N is guaranteed to run prior to the callbacks having an order value greater than N . For callbacks having the same order value, they are run in the order they were registered. It is not possible to control the registration order for two callbacks declared into two separate plugin directories so when the order is important, specify appropriate order values.

- -static_arguments *list_of_argument_values*

Specifies a list of argument values to pass to the proc in front of the normal arguments being passed.

This feature is useful to reuse a common proc for two different callback features. You use the values specified in this list to specialize your proc. In the feature_identifier section above, the list of arguments the proc must have for each feature name is documented.

For example, the argument list of the proc associated to the process_dft_specification.post_insertion feature is {root_wrapper args}. If you specified an static argument list with two values, you would need to declare two new variable in front of the root_wrapper argument as follow {static_arg1 static_arg2 root_wrapper args}. The actual name of the argument is not important but their position and count must match the specified list_of_argument_values.

Examples

This example below registers a proc to run before the system mode transition. When it detects that the context is dft and the transition is from setup to analysis, it deletes the existing Design(*design_name,design_id*) wrapper if it already exists and creates a new empty one. With this callbacks, the wrapper is guaranteed to exist and be empty when entering the analysis system mode in dft context. For this code to be available whenever Tessent Shell runs, it is included in the file *tcl_modules/your_company_name/any_name.tcl* file in one of your [plugin directory](#).

```
proc reset_design_wrapper { from_mode to_mode args } {
    if {[get_context] eq "dft" && $from_mode eq "setup" && $to_mode eq "analysis"} {
        set cur_design [get_single_name [get_current_design]]
        set design_id [get_context -design_id]
        if {[get_config_value Design("$cur_design","$design_id") -exists]} {
            delete_config_element Design("$cur_design","$design_id")
        }
        add_config_element Design("$cur_design","$design_id")
    }
}
register_callback system_mode_transition.pre reset_design_wrapper
```

register_drc

Context: dft, patterns, unspecified

Mode: setup

A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the [analyze_drcViolation](#) command and the Tesson DFTVisualizer GUI.

Usage

```
register_drc name -class class_name -short_help short_help
  -drc_proc drc_proc -context_list context_list
  [-summary_text summary_text]
  [-severity {error | warning | note | ignore}]
  [-lowest_downgradable_severity {error | warning | note | ignore}]
  [-verbose {auto | on | off}] [-rtl_synthesis_proc rtl_synthesis_proc]
  [-runs_after drc_name] [-allow_auto_fix]
```

Description

A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the [analyze_drcViolation](#) command and the Tesson DFTVisualizer GUI. Tesson Shell includes several high-level commands such as [trace_flat_model](#) and [set_attribute_value](#) giving you the ability to code in a few lines of Tcl code very complex DRC rules that run quickly and perform arbitrary complex custom checks. A comprehensive set of examples is provided below. The examples describe the complete implementation of four different DRC rules and provide the complete source code to serve as the basis for your own custom rules.

Arguments

- ***name***

A required string that will be appended to the DRC class name to make up a unique name for the DRC. Each registered DRC must have a unique <class_name><name> value. You typically use an integer as the name, and a string that identifies your company name and DRC type in the class name.

For example, the DRC rules presented in the examples below have a class name of “XYZ_S” and a name from 1 to 4. The complete DRC names are therefore “XYZ_S1”, “XYZ_S2”, “XYZ_S3”, and “XYZ_S4”. It is important to use a suffix like “XYZ_” in your class names to make sure you do not collide with built-in DRC names which do not have such prefixes. For example, even if DRC C27 does not yet exist as a built-in DRC name, it is likely it will exist in the future and your plugin would start failing when you load it into the newer version of Tesson Shell.

By default, the order of execution of the user-defined DRC rules is extracted by doing a numerical sort on the DRC name. To precisely control the order of execution, use the *-runs_after* switch described below.

- **-class *class_name***

A required switch-string pair that identifies the class name in which the DRC belongs. You must pre-register the *class_name* using the [register_drc_class](#) command.

- **-short_help *short_help***

A required switch-string pair that provides a one line description of the DRC rule. This string is used to identify the DRCs in the [DRC Violation Browser](#) of the GUI and when the [report_drc_rules](#) command is called. See the examples below for an illustration of the wording you would typically use.

- **-drc_proc *drc_proc***

A required switch-string pair defining the name of a proc that is to be called to perform the DRC. As is described in [Example 1](#) of the [plugin directory](#) section, you use the following:

```
-drc_proc [create_wrapper_proc my_tcl_proc]
```

when you are packaging your custom DRC in a plugin. The created wrapper proc adds lots of functionality to your proc such as proper Ctrl-C handling, showing the Tcl-Stack if you have a bug in your code, and most importantly loading the body of your rule the first time it is used instead of when Tesson Shell starts. The examples below illustrates the use of this switch with the use of the `create_wrapper_proc` command.

The supplied proc must accept two arguments: mode and *drc_name*. The proc will be called with the *drc_name* argument specifying the name of the DRC that called the proc. This argument is useful when you want to share a common proc for more than one DRC name. The *drc_name* value allows you to customize it for each DRC while keeping the common part common. The mode argument will have two values: enabled and run. When it has the value “enabled”, the proc must return 1 if the DRC is enable and you want it to run. Otherwise you return 0. If and only if you returned 1 will it then be called with mode equal to “run”. See the examples below for a suggested way to establish if a DRC is enabled or not.

Another way to control when the DRC runs is to specify a default severity of ignore using the -severity switch described below. You then use `set_drc_handling drc_name {error | warning | note}` when you want it to run.

- **-context_list *context_list***

A required switch-string pair that is used to define in which context the registered DRC is to be run. You typically will use “dft -no_sub_context” which means the DRC will only be requested to run in the DFT context when no sub-context (for example, -scan or -test_points) was specified. If you choose to enable the DRC in some other contexts, you may not have the information needed for it. The XYZ_S3 DRC shown in [Example 3](#) below depends on the attributes populated by running the [DFT_C6](#) DRC. DFT_C6 is only allowed in dft context with no sub context specified. It would therefore not be possible to activate XYZ_S3 in any other sub-context as the information computed by DFT_C6 would not be available.

- **-summary_text *summary_text***

An optional switch-string pair that specifies a string that is used when a DRC summary line is echoed to reflect the presence of violations for the given DRC. By default, DRC rules with severity warning or below only echo to the summary line if there was at least one violation. If you do not specify this switch, then the **short_help** string will also be used as the summary text, which is what is done in the examples shown below and in most built-in DRCs.

- **-severity error | warning | note | ignore**

An optional switch-value pair specifying the default severity the DRC is to have when enabled. If the **-lowest_downgradable_severity** is specified with a lower value, then the severity can be downgraded in your dofile using the **set_drc_handling** command. It is always possible to increase the severity of a DRC. When unspecified, the severity defaults to “error”.

A way to create a DRC that never runs by default but can be activated with the **set_drc_handling** command is to register the DRC with a default severity of “ignore”. When you want to activate the DRC, you use the “**set_drc_handling *drc_name severity***” command to bring it up to the required severity.

- **-lowest_downgradable_severity error | warning | note | ignore**

An optional switch-value pair specifying the lowest severity the DRC can be downgraded to. When unspecified, the lowest downgrade value is the default severity specified with the **-severity** switch. The lowest downgrade value cannot be higher than the default severity value specified by the **-severity** switch.

Note that the severity of a DRC can always be upgraded. For example, even if the default severity is “warning”, you can increase the severity to “error” using the **set_drc_handling** command.

- **-verbose auto | on | off**

An optional switch that controls the verbosity of the DRC. When set to “auto”, the DRC will generate one Error message per violation if the severity is “error”. Only a summary line showing the number of violations for the given DRC will be issued for severity warning or below. Setting **-verbose** to “on” will cause a note or warning to be issued for each violation even if the severity is below “error”. Setting **-verbose** to “off” will suppress the individual error message for each violation and only a summary line with the violation count will be issued, even if the severity is “error”. To view the individual violations, use the “**report_drc_rules *drc_name***” to obtain the list of violations for the given DRC name.

- **-rtl_synthesis_proc *rtl_synthesis_proc***

An optional switch-value pair that specifies the name of a proc that will be called to determine if the DRC requires anything to be synthesized in order for the DRC to check the logic. As with the DRC proc, the RTL synthesis proc has two arguments: mode and **drc_name**. Just like with the DRC proc, the RTL synthesis proc will be called with the **drc_name** argument specifying the name of the DRC that called the proc. This argument is useful when you want to share a common proc for more than one DRC name. In [Example 1](#)

below, you can see that this technique was used and a common proc called “xyz_synth.tcl” was used for all four XYZ_S DRC rules.

As with the DRC proc, it is best to use the create_wrapper_proc to encapsulate your proc as follows:

```
-rtl_synthesis_proc [create_wrapper_proc my_tcl_proc2]
```

The mode argument has multiple values. The values and their meaning are described in [Table 5-5](#).

Table 5-5. Mode values for the RTL Synthesis Proc

Mode Value	Meaning
enabled	In this mode, the proc returns 1 or 0 to indicate if it requires anything to be synthesized or not.
complete_start_module	In this mode, the proc returns a collection of modules that needs to be synthesized downward. All modules below the returned modules will also be synthesized unless the lower module is returned in the specified “complete_stop_module” collection. It must return a null string when nothing needs to be synthesized completely. You typically return the current_design when doing DRC that deals with flip-flops because until a module is synthesized, it is not possible to know if it has flip-flops or not. The DRC XYZ_S3 in Example 3 below is an example of a DRC that requires full synthesis. When you are checking connections between top-level port or pins of instances, you do not need to synthesize everything; instead, you use the selective modes described below to request synthesizing logic in the fanin or fanout of specific nodes.
complete_stop_module	In this mode, the proc returns a collection of modules that stop the downward synthesis from the specified “complete_start_module” collection. For example, if you only want to see flip-flops in the top module, you would specify the current design as the “complete_start_module” and the modules of the instances found in the current_design as the “complete_stop_module”. The code to create the “complete_stop_module” collection would be “get_modules -of_instances [get_instances * -of_type design]”.

Table 5-5. Mode values for the RTL Synthesis Proc (cont.)

Mode Value	Meaning
selective_fanin	In this mode, the proc returns a collection of pins or ports for which the fanin must be synthesized. For each node in the collection, the fanin is traced until either a net with no source, a top-level PI, or a sequential element is reached. If a net that is part of an RTL expression is reached, its associated module will be synthesized, and the above tracing process will be recursively repeated. The selective_fanin_stop_equation described next is another condition that stops the tracing.
selective_fanin_stop_equation	In this mode, the proc returns a string that represent a filtering equation based on port and pin attributes. When the equation evaluation is true, the fanin traversal is stopped at this point. The usage of such equation is illustrated in the RTL synthesis proc shown in Example 1 .
selective_fanout	In this mode, the proc returns a collection of pins or ports for which their fanout must be synthesized. For each node in the collection, the fanout is traced until either a net with no destination, a top-level PO, or a sequential element is reached. If a net that is part of an RTL expression is reached, its associated module will be synthesized, and the above tracing process recursively repeated. The selective_fanout_stop_equation describe next is another condition that stops the tracing.
selective_fanout_stop_equation	In this mode, the proc returns a string that represent a filtering equation based on port and pin attributes. When the equation evaluation is true, the fanout traversal is stopped.

- `-runs_after drc_name`

An optional switch-value pair that specifies the order a given DRC runs relative to the other DRC rules. You use this to create a sequential order of execution of the DRCs. When unspecified, the DRC will run after the DRC it follows when doing a numerical sort. Remember that each DRC has an independent way of specifying that it is enabled or not. Even if you specify `-runs_after DRCX`, the DRC will run after DRCX whether DRCX is enabled or not.

- `-allow_auto_fix`

An optional Boolean switch used that enables auto fix support within this DRC. When the switch is used, the command “`set_drc_handling drc_name`” supports the `-auto_fix` on | off switch. The default value is on. Use “`get_drc_handling drc_name -auto_fix`” to determine if auto fix is enabled or disabled for a specific DRC. The rest of the work is in the coding of the DRC. Under certain conditions, when auto fix is on, you can choose to implement an auto fix instead for given a violation. See [DFT_C9](#) for an example of a DRC rule that

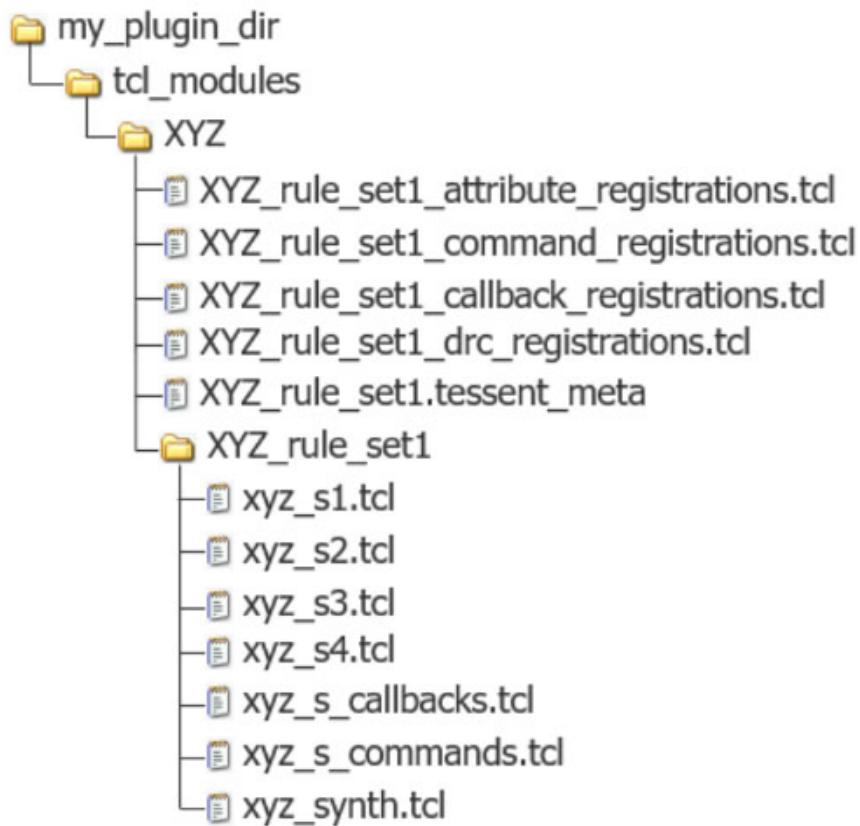
supports auto fix. Instead of giving a violation, the tool infers a DFT control point to make the set or reset sources controllable during shift.

Examples

The following examples are a set of four custom DRC rules used to provide you with guidance for coding your own rules. The four examples were chosen to cover a wide spectrum of the available infrastructure. Some of the commands and procedures described in [Example 1](#) are re-used in the other three examples.

The four DRCs rules are packaged in a *tcl_modules* plugin directory following the same organization described in the [plugin directory](#) section. The directory and files described in the examples below follow the organization presented in [Figure 5-4](#). The files located in the *XYZ* directory are loaded when Tesson Shell starts (assuming the `TESSENT_PLUGIN_PATH` environment variable points to the *my_plugin_dir* directory). The files inside the *XYZ_rule_set1* sub-directory are loaded the first time the DRC rules run. As described in the [plugin directory](#) section, you must perform the registrations and the metadata declarations in the files that are loaded into Tesson Shell at start time. The body of the procs that implement the commands, the callbacks, and the DRC rules are loaded on demand the first time the command, callback, or DRC, respectively, is run. The four example DRC rules are as follows:

- [Example 1](#) — *XYZ_S1*: All ports must go through a buffer before any other logic
- [Example 2](#) — *XYZ_S2*: No combinational path except pure feed-through paths
- [Example 3](#) — *XYZ_S3*: No inter-domain path not going through sync cells
- [Example 4](#) — *XYZ_S4*: Input pins must be sourced by output pins with consistent *xyz_function*

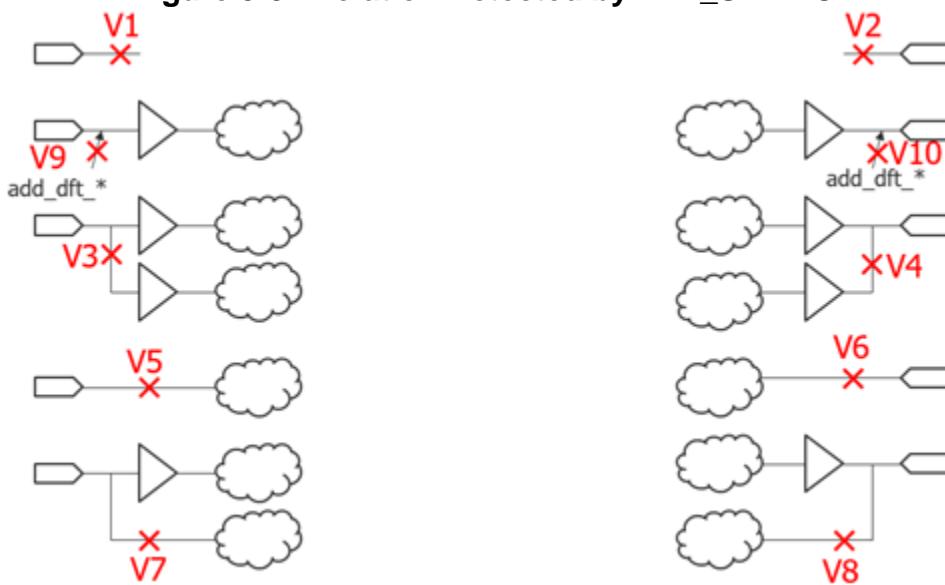
Figure 5-4. Plugin Directory Contains XYZ Custom DRC Rules

Example 1

This example implements a DRC rule that verifies that each port of the current design connects to one and only one buffer cell. Some design groups have the practice of having a single buffer connected to each port of a physical layout region such that it can be placed near the physical port of the layout region ensuring that there is no long unbuffered net inside the layout region. Some ports like a clock port are excluded automatically from this check because their buffering is handled during Clock Tree Synthesis, which happens later in the flow during layout.

The types of violation that must be detected by the DRC rule are illustrated in [Figure 5-5](#). The V1 and V2 violations detect unconnected ports. The V3 and V4 violations detect ports connected to more than one buffer cells. The V5 and V6 violations detect ports connected to logic without going through a buffer cell. The V7 and V8 violations detect ports connected to a single buffer cell but also to other logic. The V9 and V10 detects ports properly connected to a single buffer but which were the target of a [add_dft_control_points](#) or [add_dft_clock_mux](#) command. The logic is not yet inserted but after the [process_dft_specification](#) command is run, logic will be inserted, and the port will no longer be directly connected to a buffer cell.

Figure 5-5. Violation Detected by XYZ_S1 DRC



The following is the content of the *XYZ_rule_set1_command_registrations.tcl* file. It contains the registration of commands used to configure or report the settings of the XYZ_S DRC rules. The file also contains a namespace import of the `::mentor::DrcUtilities::*` procedure that will be used to facilitate the coding of the custom DRC rules. The imported commands will be described later when they are used in the DRC body description. Finally, the file declares a namespace variable that is used to store a Tcl dictionary holding the enable status as well as other options for the DRC rules.

```
#### XYZ_rule_set1_command_registrations.tcl ####
namespace eval XYZ_rule_set1 {

    ##### Command imports

    # ITCL classes
    # DrcViolation
    #
    # Procs
    # get_add_dft_node_filter
    #     returns a string used to filter gate_pin collection to know
    #     if they were the target of an add_dft_* command
    # get_add_dft_node_message
    #     returns a string explaining which add_dft_* command was
    #     issued on the node
    # get_synchronous_clock_label_groups

    namespace import ::mentor::DrcUtilities::*;

    ##### Command registration

    register_tcl_command set_xyz_s_drc_options \
        -context_list {unspecified dft patterns} \
        -system_mode_list {setup} \
        -string_option {{<enable_drc_index_range_list>} {-enable_drcs}\ \
                        {enable_drc_index_range_list}} \
        -string_option {{<disable_drc_index_range_list>} {-disable_drcs}\ \
                        {disable_drc_index_range_list}} \
        -string_option {{<excluded_ports_from_xyz_s1>} \ \
                        {-excluded_ports_from_xyz_s1}\ \
                        {excluded_ports_from_xyz_s1}} \
        -string_option {{<excluded_clocks_from_xyz_s3>} \ \
                        {-excluded_clocks_from_xyz_s3}\ \
                        {excluded_clocks_from_xyz_s3}} \
        -string_option {{<synchronizer_cells>} {-synchronizer_cells}\ \
                        {synchronizer_cells}} \
        -syntax_bnf {[<enable_drc_index_range_list>] \ \
                      [<disable_drc_index_range_list>]\ \
                      [<excluded_ports_from_xyz_s1>] \ \
                      [<excluded_clocks_from_xyz_s3>] \ \
                      [<synchronizer_cells>]} \
        -transcript tool_command \
        -tcl_proc "[create_wrapper_proc set_xyz_s_drc_options]"

    register_tcl_command report_xyz_s_drc_options \
        -context_list {unspecified dft patterns} \
        -system_mode_list {setup analysis insertion} \
        -syntax_bnf {} \
        -transcript tool_command \
        -tcl_proc "[create_wrapper_proc report_xyz_s_drc_options]"
}
```

```
# Variable definitions
variable xyz_s_drc_dict {
    drc_indexes {
        1 {
            enable 0
            excluded_ports {}
        }
        2 {
            enable 0
        }
        3 {
            enable 0
            excluded_clocks {}
        }
        4 {
            enable 0
        }
    }
    synchronizer_cells {}
}
```

The next section of code includes the content of the file *xyz_s_commands.tcl*. The file contains the body of the procs used to implement the registered commands. There is nothing specific to DRC rules in this file, and this is simply provided for completeness. You can use this example to see a use of the Tcl “dict” commands and how Tcl dictionaries are perfect to store structured data.

```
##### xyz_s_commands.tcl #####
proc report_xyz_s_drc_options {args} {
    variable xyz_s_drc_dict
    set drc_indexes_dict [dict get $xyz_s_drc_dict drc_indexes]
    puts "-----"
    puts "DRC XYZ_S* Options"
    puts "-----"
    foreach drc_index [dict keys $drc_indexes_dict] {
        set enable [dict get $drc_indexes_dict $drc_index enable]
        if {$enable} {
            puts "[string range "XYZ_S${drc_index}" " 0 8] : On"
            report_xyz_s1_excluded_ports $drc_index
            report_xyz_s1_excluded_clocks $drc_index
        } else {
            puts "[string range "XYZ_S${drc_index}" " 0 8] : Off"
        }
    }
    report_xyz_s_synchronizer_cells
    puts "-----"
}
```

```
#####
proc report_xyz_s_synchronizer_cells {} {
    variable xyz_s_drc_dict
    set synchronizer_cells [dict get $xyz_s_drc_dict synchronizer_cells]
    if {[sizeof_collection $synchronizer_cells] > 0} {
        puts "Synchronizer cells : [join [get_name_list $synchronizer_cells]
", "]"
    }
}
#####
proc report_xyz_s1_excluded_ports {drc_index} {
    variable xyz_s_drc_dict
    if {[get_resource_design_is_elaborated] && $drc_index == 1} {
        set excluded_ports [dict get $xyz_s_drc_dict \
                                drc_indexes 1 excluded_ports]
        if {[sizeof_collection $excluded_ports] > 0} {
            puts " Excluded ports:"
            puts "     [join [get_name_list $excluded_ports] "\n      "
        }
    }
}
#####
proc report_xyz_s1_excluded_clocks {drc_index} {
    variable xyz_s_drc_dict
    if {[get_resource_design_is_elaborated] && $drc_index == 3} {
        set excluded_clocks [dict get $xyz_s_drc_dict drc_indexes 3
excluded_clocks]
        if {[sizeof_collection $excluded_clocks] > 0} {
            puts " Excluded Clocks:"
            puts "     [join [get_name_list [get_design_objects -from_objects
$excluded_clocks -map_pseudo_ports]] "\n      "
        }
    }
}
#####
proc set_xyz_s_drc_options {args} {
    array set ARGS $args
    set error_status 0
    if {[info exists ARGS(enable_drc_index_range_list)]} {
        set_xyz_s_drc_enables 1 $ARGS(enable_drc_index_range_list)
    }
    if {[info exists ARGS(disable_drc_index_range_list)]} {
        set_xyz_s_drc_enables 0 $ARGS(disable_drc_index_range_list)
    }
    if {[info exists ARGS(excluded_ports_from_xyz_s1)]} {
        set_xyz_s1_drc_excluded_ports $ARGS(excluded_ports_from_xyz_s1)
    }
    if {[info exists ARGS(excluded_clocks_from_xyz_s3)]} {
        set_xyz_s3_drc_excluded_clocks $ARGS(excluded_clocks_from_xyz_s3)
    }
    if {[info exists ARGS(synchronizer_cells)]} {
        set_xyz_s_synchronizer_cells $ARGS(synchronizer_cells)
    }
    if {$error_status} {
        return -code error
    }
}
```

```
#####
proc set_xyz_s_drc_enables {value drc_index_range_list} {
    variable xyz_s_drc_dict
    upvar error_status error_status
    set drc_indexes_dict [dict get $xyz_s_drc_dict drc_indexes]
    set drc_index_list [dict keys $drc_indexes_dict]
    if {$value} {
        #Remove already enabled ones when enabling new ones.
        foreach drc_index $drc_index_list {
            dict set xyz_s_drc_dict drc_indexes $drc_index enable 0
        }
    }
    foreach drc_index_range $drc_index_range_list {
        if {[regexp {^([0-9])+(:-([0-9]+))?\$} $drc_index_range -> \
                    index0 index1]} {
            if {$index1 eq ""} {
                set index1 $index0
            }
            if {$index1 < $index0} {
                set incr_val -1
                set stop_condition "\$i >= \$index1"
            } else {
                set incr_val 1
                set stop_condition "\$i <= \$index1"
            }
            for {set i $index0} $stop_condition {incr i $incr_val} {
                if {$i in $drc_index_list} {
                    dict set xyz_s_drc_dict drc_indexes $i enable $value
                }
            }
        } else {
            display_message -error "DRC index range '$drc_index_range' does not
match # or #-#."
            set error_status 1
        }
    }
}
#####
proc set_xyz_s1_drc_excluded_ports {excluded_ports_from_xyz_s1} {
    variable xyz_s_drc_dict
    upvar error_status error_status
    if {! [get_resource design_is_elaborated]} {
        display_message "The -excluded_ports_from_xyz_s1 can only be used
after the design is elaborated."
        set error_status 1
    } else {
        set port_objects [check_current_design_port_spec
$excluded_ports_from_xyz_s1 "<excluded_ports_from_xyz_s1>"]
        dict set xyz_s_drc_dict drc_indexes 1 excluded_ports $port_objects
    }
}
```

```
#####
proc check_current_design_port_spec {current_design_port_spec
source_string} {
    upvar error_status error_status
    if {[is_collection $current_design_port_spec]} {
        if {[sizeof_collection $current_design_port_spec] == 0} {
            return {}
        }
        set non_port_objects [filter_collection $current_design_port_spec \
                               "object_type != port"]
        if {[sizeof_collection $non_port_objects] > 0} {
            display_message -error \
                "The supplied $source_string collection contains elements which are
not port objects:\n  [join [get_name_list $non_port_objects] "\n  ]"
            set error_status 1
            return {}
        }
        return $current_design_port_spec
    } else {
        set returned_collection {}
        foreach name_pattern $current_design_port_spec {
            set port_objects [get_ports [list $name_pattern] -silent]
            if {[sizeof_collection $port_objects] == 0} {
                display_message -error \
                    "The name_pattern '$name_pattern' supplied in $source_string does
not match any port on the current design."
                set error_status 1
            } else {
                append_to_collection returned_collection $port_objects
            }
        }
        return $returned_collection
    }
}
#####
proc set_xyz_s3_drc_excluded_clocks {excluded_clocks_from_xyz_s3} {
    variable xyz_s_drc_dict
    upvar error_status error_status
    if {! [get_resource design_is_elaborated]} {
        display_message "The -excluded_clocks_from_xyz_s3 can only be used
after the design is elaborated."
        set error_status 1
    } else {
        set clock_objects [check_clock_spec $excluded_clocks_from_xyz_s3 \
                           "<excluded_clocks_from_xyz_s3>"]
        dict set xyz_s_drc_dict drc_indexes 3 excluded_clocks $clock_objects
    }
}
```

```
#####
proc check_clock_spec {clock_spec source_string} {
    upvar error_status error_status
    if {[is_collection $clock_spec]} {
        if {[sizeof_collection $clock_spec] == 0} {
            return {}
        }
        set non_clock_objects {}
        foreach_in_collection object $clock_spec {
            if {[get_clock_option $object -type -silent] eq ""} {
                append_to_collection non_clock_objects $object
            }
        }
        if {[sizeof_collection $non_clock_objects] > 0} {
            display_message -error \
                "The supplied $source_string collection contains elements which are
not clock objects:\n [join [get_name_list $non_clock_objects] "\n "]"
            set error_status 1
            return {}
        }
        return $clock_spec
    } else {
        set returned_collection {}
        foreach name_pattern $clock_spec {
            set clock_objects [get_clocks [list $name_pattern] -silent]
            if {[sizeof_collection $clock_objects] == 0} {
                display_message -error \
                    "The name_pattern '$name_pattern' supplied in $source_string does
not match any clock."
                set error_status 1
            } else {
                append_to_collection returned_collection $clock_objects
            }
        }
        return $returned_collection
    }
}
#####
proc set_xyz_s_synchronizer_cells {synchronizer_cells_spec} {
    variable xyz_s_drc_dict
    upvar error_status error_status
    set synchronizer_cells_objects \
        [check_module_cell_spec $synchronizer_cells_spec \
            "<synchronizer_cells>"]
    dict set xyz_s_drc_dict synchronizer_cells $synchronizer_cells_objects
}
```

```
#####
proc check_module_cell_spec {module_cell_spec source_string} {
    upvar error_status error_status
    if {[is_collection $module_cell_spec]} {
        if {[sizeof_collection $module_cell_spec] == 0} {
            return {}
        }
        #Check that they are all modules
        set non_module_objects [filter_collection $module_cell_spec
"object_type != module"]
        if {[sizeof_collection $non_module_objects] > 0} {
            display_message -error \
                "The supplied $source_string collection contains elements which are
not module objects:\n[join [get_name_list $non_module_objects] "\n "]"
            set error_status 1
            return {}
        }
        #Now that I know they are all modules, check if they are all of
        #type cells.
        set non_cell_modules [filter_collection $module_cell_spec \
                "type != cell"]
        if {[sizeof_collection $non_cell_modules] > 0} {
            display_message -error \
                "The supplied $source_string collection contains modules which are
not of type cell:\n[join [get_name_list $non_cell_modules] "\n "]"
            set error_status 1
            return {}
        }
        return $module_cell_spec
    } else {
        set returned_collection {}
        foreach name_pattern $module_cell_spec {
            set module_objects [get_modules [list $name_pattern] -silent]
            if {[sizeof_collection $module_objects] == 0} {
                display_message -error \
                    "The name_pattern '$name_pattern' supplied in $source_string does
not match any module."
                set error_status 1
            } else {
                set non_cell_modules [filter_collection $module_objects
                        "type != cell"]
                if {[sizeof_collection $non_cell_modules] > 0} {
                    display_message -error \
                        "The name_pattern '$name_pattern' supplied in $source_string
matches modules which are not of type cell:\n[join [get_name_list
$non_cell_modules] "\n "]"
                    set error_status 1
                } else {
                    append_to_collection returned_collection $module_objects
                }
            }
        }
        return $returned_collection
    }
}
```

The next code block shows the content of the file *XYZ_rule_set1_drc_registrations.tcl*. This file includes the registration of the four XYZ_S DRC rules. The first line registers the XYZ_S DRC class using the `register_drc_class` command. Below are the four DRC rule registrations using the `register_drc` command. Notice how the `create_wrapper_proc` command is used around the referenced proc names. The wrapper proc provides a lot of built-in functionality to your proc such as proper Ctrl-C handling, dumping the stack if you have a Tcl bug in your code, and most importantly, loading the body of the procs the first time the DRC runs. Because the `create_wrapper_proc` is called inside namespace “XYZ_rule_set1”, the “*.tcl” files found inside the subdirectory by the name of the namespace will be loaded the first time the wrapper proc is called.

```
##### XYZ_rule_set1_drc_registrations.tcl #####
namespace eval XYZ_rule_set1 {
    # DRC registrations

    register_drc_class XYZ_S
    register_drc 1 \
        -class XYZ_S \
        -short_help "All ports must go through a buffer before any other
logic" \
        -context_list "dft -no_sub_context" \
        -drc_proc [create_wrapper_proc XYZ_S1_drc] \
        -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 2 \
        -class XYZ_S \
        -short_help "No combinational path except pure feed-through paths" \
        -context_list "dft -no_sub_context" \
        -drc_proc [create_wrapper_proc XYZ_S2_drc] \
        -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 3 \
        -class XYZ_S \
        -short_help "No inter-domain path not going through sync cells" \
        -context_list "dft -no_sub_context" \
        -drc_proc [create_wrapper_proc XYZ_S3_drc] \
        -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 4 \
        -class XYZ_S \
        -short_help "Input pins must be sourced by output pins with
consistent xyz_function" \
        -context_list "dft -no_sub_context" \
        -drc_proc [create_wrapper_proc XYZ_S4_drc] \
        -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]
}
```

The next code block section shows the content of the *xyz_s1.tcl* file which contains the body definition of the XYZ_S1 DRC rule. The first proc is the main proc that has the mode and drc_name arguments. When called with mode equal to “enabled”, the proc returns the enable flag of the XYZ_S1 DRC found in the *xyz_s_drc_dict*. The command used to populate the

dictionary was presented above in the second code block of [Example 1](#) and is found in file *xyz_s_commands.tcl*. The proc that actually checks if any of the situations described in [Figure 5-5](#) are present is the second proc named “XYZ_S1_check_port”. When this proc detects an error condition, it returns a Tcl dictionary with the following keys: “path_list0”, “path_list1”, and “msg_list”. The returned value is then used to add the DRC violation with the [add_drcViolation](#) command. As stated in the [add_drcViolation](#) command section, it requires one argument, which is an object of an ITCL class having a prescribed set of methods. A complete ITCL class definition is already provided in the tool, and you typically will never need to define your own as the definition is very flexible and should be enough to cover most of your needs. See the [add_drcViolation](#) command section for a detailed description of this ITCL class and the method it provides to configure it and to service the DRC commands.

You create a new object of the `DrcViolation` class using the following command:

```
set violation [DrcViolation #auto]
```

You use the “dict_set” method of the object stored in the “violation” variable to configure the object. Once configured, you add the object as a violation using the following command:

```
add_drcViolation [namespace current] ::$violation
```

This DRC is not doing complex tracing, and the tracing does not need to traverse cells, nor does the tracing need to consider simulation values. This is why the “XYZ_S1_check_port” proc makes use of the [get_fanins](#) and [get_fanouts](#) commands instead of the [trace_flat_model](#) command. In such simple situation, using the [get_fanins](#)/[get_fanouts](#) is better because the commands operate on the hierarchical data model. If you have a port that goes down through several levels of hierarchy but ends up having no fanout to a cell, then the port will be seen as having no destination in the flat model; however, because this DRC is using [get_fanouts](#), the violation will highlight the path to the internal pin, which has no further connection. In more complex DRC like the other three examples, using the [trace_flat_model](#) command is needed because the DRCs involve tracing through multiple levels of logic and must consider a simulation context.

```

##### xyz_s1.tcl #####
proc XYZ_S1_drc {mode drc_name} {
    variable xyz_s_drc_dict
    if {$mode eq "enabled"} {
        return [dict get $xyz_s_drc_dict drc_indexes 1 enable]
    }
    set excluded_ports [dict get $xyz_s_drc_dict drc_indexes 1 \
                           excluded_ports]
    # Ports defined as clocks are auto excluded
    append_to_collection excluded_ports \
        [filter_collection [get_clocks -silent] "object_type==port"]
    foreach direction {forward backward} {
        if {$direction eq "forward"} {
            set ports_to_trace \
                [remove_from_collection [get_ports -direction input] \
                                         $excluded_ports]
        } else {
            set ports_to_trace \
                [remove_from_collection [get_ports -direction output] \
                                         $excluded_ports]
        }
        foreach_in_collection port_to_trace $ports_to_trace {
            set return_dict [XYZ_S1_check_port $port_to_trace \
                                         direction $direction]
            if {[dict size $return_dict] > 0} {
                # There was a violation
                set violation [DrcViolation #auto]
                set path_list0 [dict get $return_dict path_list0]
                $violation dict_set list_of_paths side0 $path_list0
                $violation dict_set path_direction side0 $direction
                if {[sizeof_collection $path_list0] > 1} {
                    $violation dict_set end_callout_string side0 \
                        "Invalid connection"
                }
                if {[dict exists $return_dict path_list1]} {
                    set path_list1 [dict get $return_dict path_list1]
                    $violation dict_set list_of_paths side1 $path_list1
                    $violation dict_set path_direction side1 $direction
                    if {[sizeof_collection $path_list1] > 1} {
                        $violation dict_set end_callout_string side1 \
                            "Second invalid connection"
                    }
                }
                $violation dict_set message_string_list \
                    [dict get $return_dict msg_list]
                $violation dict_set start_callout_string side0 \
                    "Port with XYZ_S1 violation."
                add_drcViolation [namespace current]::$violation
            }
        }
    }
}

```

```
#####
proc XYZ_S1_check_port {port args} {
    variable xyz_s_drc_dict
    array set ARGS {direction forward}
    array set ARGS $args
    set direction $ARGS(direction)
    set msg [XYZ_S1_check_object_has_no_add_dft_logic $port]
    if {[string length $msg] > 0} {
        dict lappend return_dict msg_list \
            "The port '[get_single_name $port]' $msg"
        dict lappend return_dict path_list0 $port
        return $return_dict
    }
    if {$direction eq "forward"} {
        set objects [get_fanouts $port -stop_on cell_pin]
    } else {
        set objects [get_fanins $port -stop_on cell_pin]
    }
    # Filter out ports and nets in fanout or fanin
    set pins [filter_collection $objects "object_type==pin"]
    if {[sizeof_collection $pins] == 0} {
        # There are no pins
        dict lappend return_dict msg_list \
            "The port '[get_single_name $port]' is not connected to a pin."
        dict lappend return_dict path_list0 $port
        return $return_dict
    }
    if {[sizeof_collection $pins] > 1} {
        # There are more than one pin
        dict lappend return_dict msg_list \
            "The port '[get_single_name $port]' is connected to
' [sizeof_collection $pins]' pins:"
        set cnt 1
        set max 3
        foreach name [get_name_list $pins] {
            if {$cnt > $max} {
                dict lappend return_dict msg_list "... "
                break
            }
            dict lappend return_dict msg_list " $name"
            incr cnt
        }
        #Show the first two in the schematic
        dict lappend return_dict path_list0 \
            [add_to_collection $port [index_collection $pins 0]]
        dict lappend return_dict path_list1 \
            [add_to_collection $port [index_collection $pins 1]]
        return $return_dict
    }
    # There is one pin
    # check if it was not the target of a add_dft_* command
    set msg [XYZ_S1_check_object_has_no_add_dft_logic $pins]
    if {[string length $msg] > 0} {
        dict lappend return_dict msg_list \
```

```

"The port '[get_single_name $port]' is connected to a pin \
'[get_single_name $pins]'"
dict lappend return_dict msg_list "which ${msg}"
dict lappend return_dict path_list0 [add_to_collection $port $pins]
return $return_dict
}
set instance [get_instances -of_pins $pins]
set simulation_function \
[get_attribute_value_list $instance -name simulation_function]
if {$simulation_function ne "buffer"} {
    #Not a pin of a buffer cell
    dict lappend return_dict msg_list \
"The port '[get_single_name $port]' is connected to a pin \
'[get_single_name $pins]'"
    dict lappend return_dict msg_list \
"which is not a pin of an instance with simulation_function equal to
'buffer'."
    dict lappend return_dict path_list0 [add_to_collection $port $pins]
    return $return_dict
}
return {}
}

#####
# This proc check if the add_dft_node_filter evaluates true on the object.
# If it does, the object was the target of an add_dft_* command
# The return list explains which one.
# If it does not, an empty list is returned
#
proc XYZ_S1_check_object_has_no_add_dft_logic {object} {
    set object_with_add_dft_logic [filter_collection $object \
"[get_add_dft_node_filter hier] || dft_clock_mux_destinations"]
    if {[sizeof_collection $object_with_add_dft_logic] > 0} {
        return [get_add_dft_node_message $object_with_add_dft_logic hier]
    } else {
        return [list]
    }
}

```

Below is the source code used to request various parts of the design to be synthesized based on which DRC was enabled. The “drc_name” argument is used to decide what needs to be synthesized or not. The XYZ_S1, XYZ_S2, and XYZ_S4 DRC rules use the selective approach because they check the fanin and fanout of ports and pins. The XYZ_S3 DRC uses the complete synthesis methods because the DRC checks for inter-domain paths between sequential elements, and the sequential elements can only be identified after complete synthesis. Refer to the *-rtl_synthesis_proc rtl_synthesis_proc* argument section above for more details about the various argument values for this proc.

```
##### xyz_synth.tcl #####
proc XYZ_S_synth {mode drc_name args} {
    variable xyz_s_drc_dict
    set drc_index [string range $drc_name 5 end]
    switch -exact $mode {
        "enabled" {
            return [dict get $xyz_s_drc_dict drc_indexes $drc_index enable]
        }
        "complete_start_module" {
            if {$drc_name in {XYZ_S3}} {
                return [get_current_design]
            }
        }
        "complete_stop_module" {
        }
        "selective_fanin" {
            if {$drc_name in {XYZ_S1}} {
                return [get_xyz_s1_selective_fanin_objects output]
            }
            if {$drc_name in {XYZ_S2}} {
                return [get_ports -direction output]
            }
            if {$drc_name in {XYZ_S4}} {
                return [get_xyz_s4_selective_fanin_objects]
            }
        }
        "selective_fanin_stop_equation" {
            if {$drc_name in {XYZ_S4}} {
                return "xyz_function"
            }
        }
        "selective_fanout" {
            if {$drc_name in {XYZ_S1}} {
                return [get_xyz_s1_selective_fanin_objects input]
            }
        }
        "selective_fanout_stop_equation" {
        }
    }
    return {}
}
#####
proc get_xyz_s1_selective_fanin_objects {direction} {
    variable xyz_s_drc_dict
    set excluded_ports [filter_collection [get_clocks -silent] \
        "object_type == port"]
    append_to_collection excluded_ports \
        [dict get $xyz_s_drc_dict drc_indexes 1 excluded_ports] \
        -unique
    return [remove_from_collection [get_ports -direction $direction] \
        $excluded_ports]
}
```

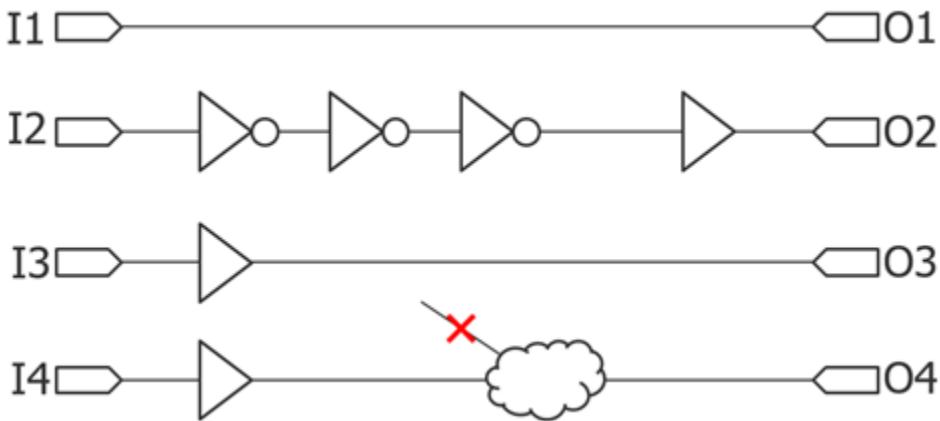
```
#####
proc get_xyz_s4_selective_fanin_objects {} {
    set attributed_objects \
        [get_attributed_objects -attribute_name xyz_function \
                                -object_types {port pin}]
    return [filter_collection $attributed_objects \
            "object_type == port && direction == output || \
            object_type == pin && direction == input"]
}
```

Example 2

This example implements a DRC rule that verifies that the design does not have combinational path between input and output ports unless they are pure feed-through paths containing only single-input cells such as inverters and buffers. [Figure 5-6](#) shows the kind of paths that need to be allowed or rejected. The example DRC rule only detects that a path has no multi-input gate along the way. If you also run XYZ_S1 presented in [Example 1](#), you would reject the path from I1 to O1 because it has no buffer. You could also chose to write another DRC to make sure there is no inversion in the feed-through paths if this is one of your requirements. You would do this after you have found the path meeting XYZ_S2. You would use [trace_flat_model](#) with the “-controllability controlling” switch and use the “-store_polarity_to_tagged_objects” switch to store the polarity of the paths.

In the XYZ_S2 DRC, the [trace_flat_model](#) command is used to trace the connected fanin of every output ports and tag any path that reaches an input port of the current design. A simple [filter_collection](#) command using the “gate_input_count” [Gate_pin](#) attribute is used to detect if any multi-input gate exists along the path between the PO and the PI. As you can see below, the code to implement this DRC is very simple and fits in a page of code.

Figure 5-6. Combinational Paths Allowed and Rejects by XYZ_S2



```
##### xyz_s2.tcl #####
proc XYZ_S2_drc {mode drc_name} {
    variable xyz_s_drc_dict
    if {$mode eq "enabled"} {
        return [dict get $xyz_s_drc_dict drc_indexes 2 enable]
    }
    set output_ports [get_ports -direction output -silent]
    foreach_in_collection output_port $output_ports {
        set hits [trace_flat_model \
                    -from $output_port \
                    -direction backward \
                    -controllability connected \
                    -stop_at_first_tag on \
                    -tag_condition "primitive_name==PI" \
                    -store_paths_to_tagged_objects paths_to_tagged_objects]

        set path_to_tagged_objects [lindex $paths_to_tagged_objects 0]
        set multi_input_gates [filter_collection $path_to_tagged_objects \
                                "gate_input_count > 1 && direction == output"]
        set multi_input_cells {}
        set msg_list [list]
        if {[sizeof_collection $multi_input_gates] > 0} {
            append_to_collection multi_input_cells \
                [get_instances -of_gate_pins $multi_input_gates -silent] -unique
            set multi_input_cell_count [sizeof_collection $multi_input_cells]
            set violation [DrcViolation #auto]
            $violation dict_set list_of_paths side0 \
                [list $path_to_tagged_objects]
            $violation dict_set path_direction side0 backward
            $violation dict_set start_callout_string side0 \
                "PO with path to PI with multi input gates along the path"
            $violation dict_set end_callout_string side0 "PI in fanin of PO"
            set start_port [index_collection $path_to_tagged_objects 0]
            set end_port [index_collection $path_to_tagged_objects \
                            [expr [sizeof_collection $path_to_tagged_objects] - 1]]
            lappend msg_list "The output port '[get_single_name $start_port]' \
has a combinational path from the input port '[get_single_name \
$end_port]'"
            lappend msg_list "but the path is not a pure feed-through as the path \
crosses $multi_input_cell_count multi-input cells:"
            set cnt 1
            set max 3
            foreach_in_collection multi_input_cell $multi_input_cells {
                if {$cnt > $max} {
                    lappend msg_list "... "
                    break
                }
                lappend msg_list " [get_single_name $multi_input_cell]"
                incr cnt
            }
            $violation dict_set message_string_list $msg_list
            add_drcViolation [namespace current]::$violation
        }
    }
}
```

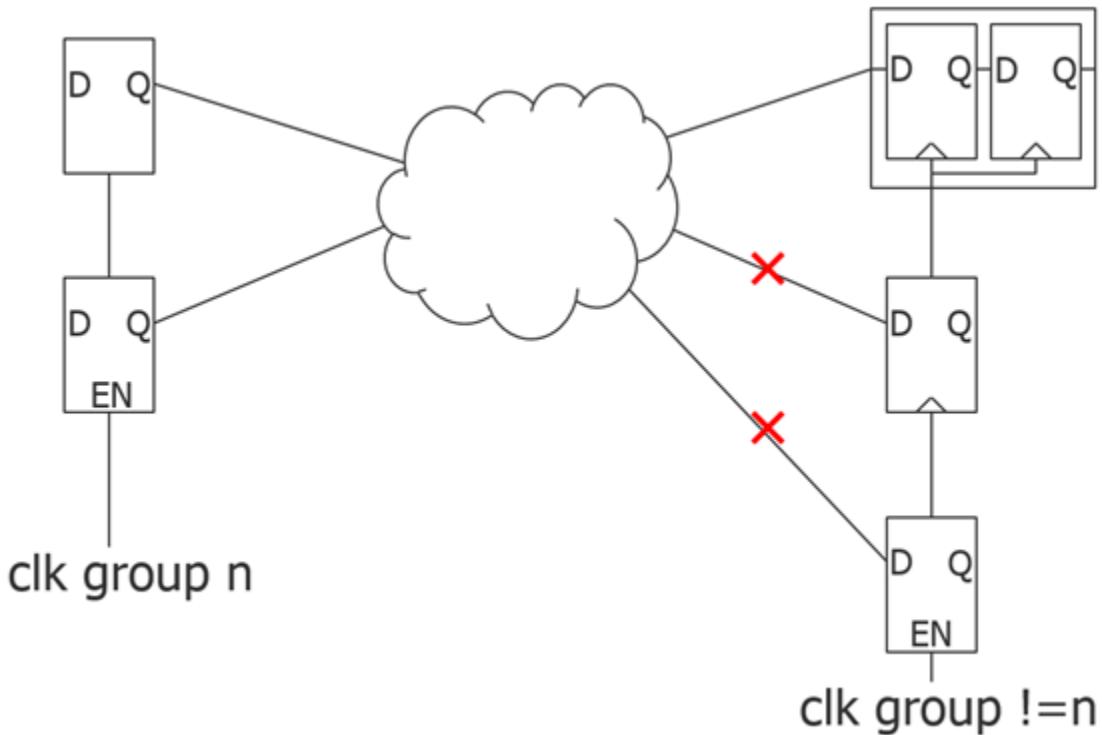
Example 3

This example implements a DRC rule that verifies there is no path from a sequential cell on a given synchronous clock group that reaches a sequential cell on another synchronous clock group unless the destination sequential cell is a synchronizer cell. The types of violations that are to be found are illustrated in [Figure 5-7](#). The implementation of this DRC is very simple once DFT_C6 has run and properly set the `clock_domain_label` attribute on each flip-flops. The implementation is shown in the example. You can see that it creates a collection of all DFF Q gate_pins. It then filters them per synchronous clock groups and issue one `trace_flat_model` command per groups. The hit equation is any DFF or DLAT gate_pin on a different synchronous clock group. The “xyz_exclude” attribute is used to tag the inputs of the synchronizer cells and any gate_pin of a sequential cell belonging to an excluded clock domain. If you go back to the example code block describing the content of the file `XYZ_rule_set1_command_registrations.tcl` in [Example 1](#), you will see that the “`set_xyz_s_drc_options`” command was equipped with an `-excluded_clocks_from_xyz_s3` switch to allow excluding some clock domains from the check. For example, it is common to allow interactions between the IJTAG static signal (TCK domain) and the functional logic without going through a synchronizer cell. The “xyz_exclude” attribute was registered in the file `XYZ_rule_set1_attribute_registrations.tcl` file, and its content is shown below.

Finally, notice that the `trace_flat_model` command is used to do a forward walk. [Figure 5-8](#) on page 1342 and [Figure 5-9](#) on page 1342 illustrate the difference between using a forward versus a backward walk. The `trace_flat_model` command is very efficient and never traverses the same gate twice in a given trace. You would be visiting each gate potentially thousands of times if you traced through a gate as part of all possible paths it belongs to. When you use a backward trace, you are guaranteed to find all hits in the fanin, but not all the sources that may reach the hits. The same is true in the other direction. In this example DRC, we are interested in finding all the destination gates sourced by at least one sequential cell on a different domain. We do not need to find all the source elements, just the failing destinations which is why a forward tracing is used. If you wanted to find all elements sourcing a different domain and all elements receiving from a different domain, you would need to use two traces, one forward and one backward. The backward trace would identify the TX flops and the forward trace would identify the RX flop.

This DRC example also makes use of the `register_callback` command to implement a check when `check_design_rules` is called to make sure the synchronizer cell was defined using the `set_xyz_s_drc_options` command and that the DFT_C6 rule was turned on. The files to register the callback and the one containing the body definition are listed at the end of this example. Notice that DFT_C6 is enabled by using the “`set_dft_specification_requirements -logic_test on`” command. That also enables DFT_C9. If you don’t want to have to worry about the asynchronous Set and Reset issues to run this DRC, you can simply disable DFT_C9 using “`set_drc_handling dft_c9 ignore`”.

Figure 5-7. Violation Detected by XYZ_S3



```
##### xyz_s3.tcl #####
proc XYZ_S3_drc {mode drc_name} {
    variable xyz_s_drc_dict
    if {$mode eq "enabled"} {
        return [dict get $xyz_s_drc_dict drc_indexes 3 enable]
    }
    set excluded_clocks [dict get $xyz_s_drc_dict drc_indexes 3
excluded_clocks]
    set sync_clock_label_groups [get_synchronous_clock_label_groups \
                                -excluded_clocks $excluded_clocks]
    reset_attribute_value -object_type gate_pin -name xyz_exclude
    set synchronizer_modules [dict get $xyz_s_drc_dict synchronizer_cells]
    if {[sizeof_collection $synchronizer_modules] > 0} {
        set synchronizer_instances
            [get_instances -of_modules $synchronizer_modules -silent]
        if {[sizeof_collection $synchronizer_instances] > 0} {
            set synchronizer_cell_pins \
                [get_pins -of_instances $synchronizer_instances]
            set_attribute_value \
                [get_gate_pins -of_pins $synchronizer_cell_pins] -name xyz_exclude
        }
    }
}
```

```

set seq_q [get_gate_pins -of_gate_type sequential \
            -filter {primitive_port_name==Q}]

set_current_simulation_context stable_after_setup

foreach sync_clock_label_group $sync_clock_label_groups {
    set equal_list [list]
    set not_equal_list [list]
    # Example sync_group =A, excluded_clocks = TCK
    # Start from all SEQ with "clock_domain_label == A"
    # Tag all seq element with "clock_domain_label != A &&
    #                               clock_domain_label!= TCK"
    foreach label $sync_clock_label_group {
        lappend equal_list "clock_domain_label == $label "
        lappend not_equal_list "clock_domain_label != $label "
    }
    set filter      [join $equal_list ||]
    if {[sizeof_collection $excluded_clocks] > 0} {
        foreach_in_collection excluded_clock $excluded_clocks {
            lappend not_equal_list "clock_domain_label != [get_clock_option
$excluded_clock -label] "
        }
    }
    set tag_condition [join $not_equal_list &&]
    #Stop on xyz_exclude to not traverse data pins of synchronizer cells
    set hits [trace_flat_model -from [filter_collection $seq_q $filter] \
              -direction forward \
              -controllability unblocked \
              -stop_condition "xyz_exclude" \
              -tag_condition "primitive_name == DFF && \
                              primitive_port_name == D && \
                              $tag_condition && !xyz_exclude" \
              -store_paths_to_tagged_objects paths_to_tagged_objects]
    foreach path_to_tagged_objects $paths_to_tagged_objects {
        set msg_list [list]
        set mapped_path_to_tagged_objects \
            [filter_collection $path_to_tagged_objects fault_site]
        set start_gate_pin [index_collection $path_to_tagged_objects 0]
        set start_clock_domain_label \
            [get_attribute_value_list $start_gate_pin \
             -name clock_domain_label]
        set end_gate_pin [index_collection $path_to_tagged_objects \
            [expr [sizeof_collection $path_to_tagged_objects] - 1]]
        set end_clock_domain_label \
            [get_attribute_value_list $end_gate_pin \
             -name clock_domain_label]
        set mapped_start_name [get_single_name [index_collection \
            $mapped_path_to_tagged_objects 0]]
        set mapped_end_name [get_single_name [index_collection \
            $mapped_path_to_tagged_objects \
            [expr [sizeof_collection $mapped_path_to_tagged_objects] - 1]]]
        set violation [DrcViolation #auto]
        $violation dict_set simulation_context \
            [get_current_simulation_context]
        $violation dict_set list_of_paths side0 \
            [list $path_to_tagged_objects]
        $violation dict_set path_direction side0 forward
        $violation dict_set start_callout_string side0 "Source sequential

```

```
cell on domain label '$start_clock_domain_label'
    $violation dict_set end_callout_string side0 "Destination sequential
cell on domain label '$end_clock_domain_label'
    lappend msg_list "The sequential cell '$mapped_end_name' which is
not a synchronizer cell is on clock domain label
'$end_clock_domain_label'.
    lappend msg_list "It has sequential cell '$mapped_start_name' on
clock domain label '$start_clock_domain_label' in its fanin"
    lappend msg_list "which is not a synchronizer cell."
    $violation dict_set message_string_list $msg_list
    add_drcViolation [namespace current]:::$violation
}
}
}

##### XYZ_rule_set1_attribute_registrations.tcl #####
namespace eval XYZ_rule_set1 {

    ### Attribute registrations
    register_attribute -name xyz_function \
        -object_type {port pin gate_pin} \
        -value_type string \
        -default ""

    set_attribute_options -name xyz_function \
        -object_type {port pin} \
        -preserve_boundary_in_flat_model on \
        -display_in_gui on

    register_attribute -name xyz_exclude \
        -object_type gate_pin \
        -value_type boolean
}

}
```

Figure 5-8. Forward Walk to Find All Destination Hits

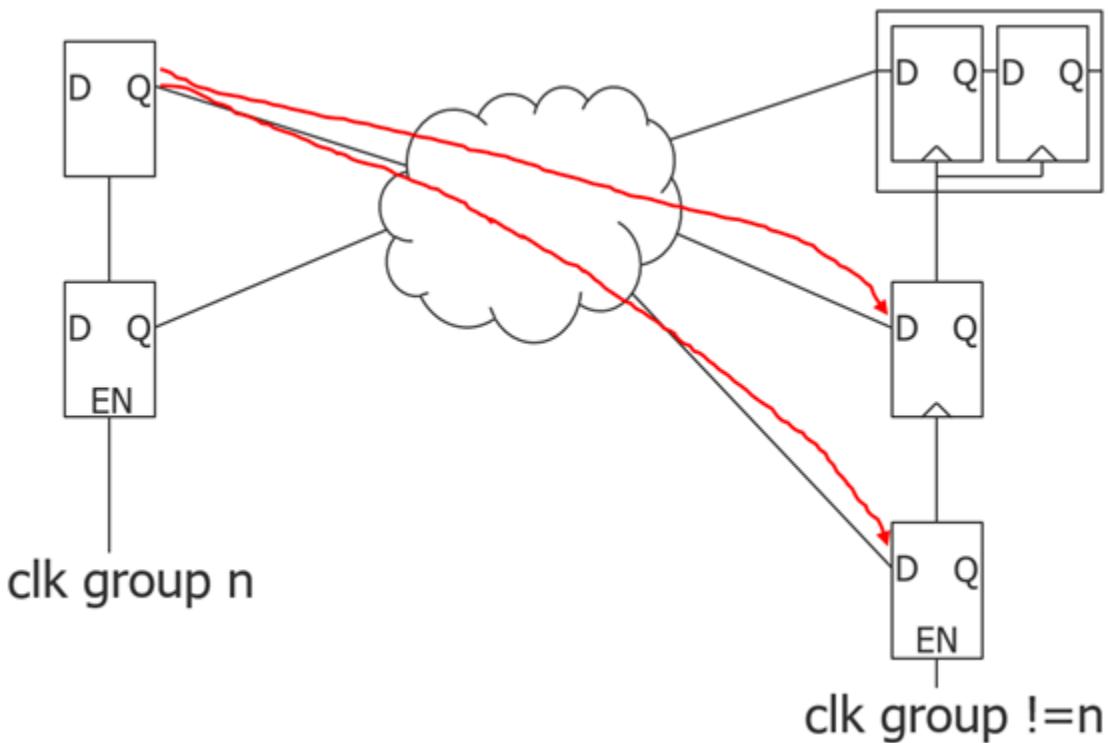
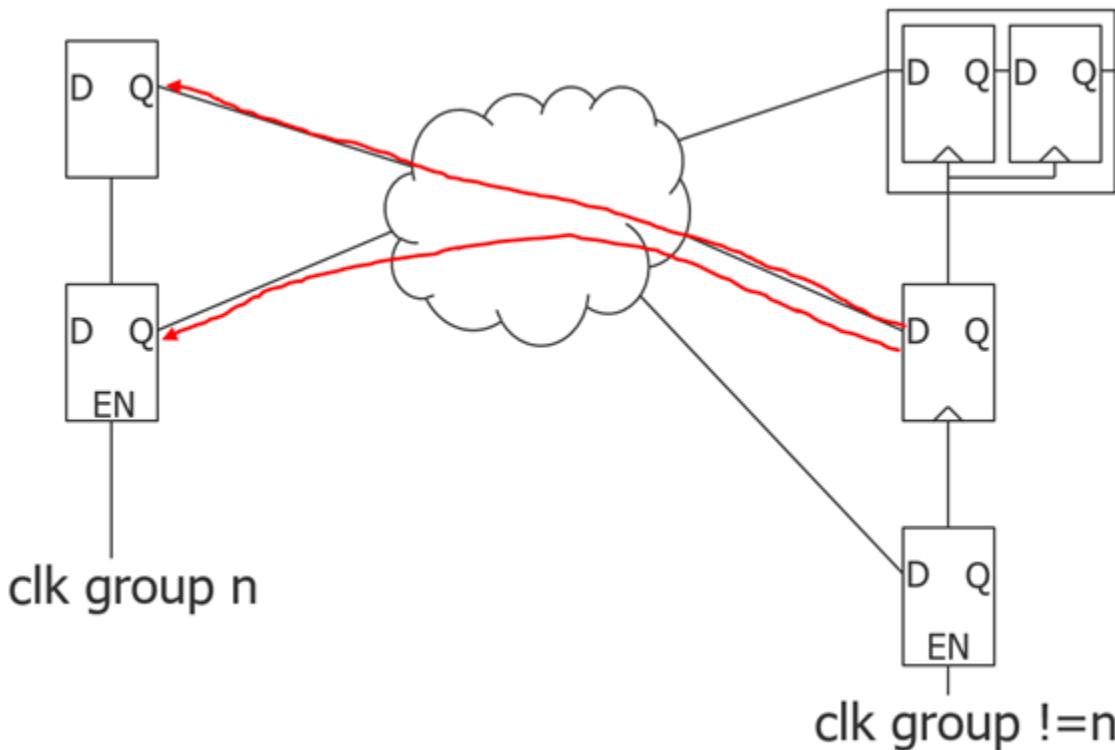


Figure 5-9. Backward Walk to Final All Source Hits



```
##### XYZ_rule_set1_callback_registrations.tcl #####
namespace eval XYZ_rule_set1 {
    register_callback elaboration_creation.post \
        [create_wrapper_proc clear_xyz_s_drc_options]
    register_callback system_mode_transition.pre \
        [create_wrapper_proc check_xyz_s3_requirements]
}

#####
# xyz_s_callbacks.tcl #####
# elaboration_creation.post
proc clear_xyz_s_drc_options {args} {
    variable xyz_s_drc_dict
    variable xyz_s_drc_reset_dict
    dict set xyz_s_drc_dict drc_indexes 1 excluded_ports {}
    dict set xyz_s_drc_dict drc_indexes 3 excluded_clocks {}
    dict set xyz_s_drc_dict synchronizer_cells {}
}
# system_mode_transition.pre
proc check_xyz_s3_requirements {from to args} {
    variable xyz_s_drc_dict
    set error_status 0
    if {$from eq "setup" && $to eq "analysis" && [get_context] eq "dft" && \
        [get_context -no_sub_context] && \
        [dict get $xyz_s_drc_dict drc_indexes 3 enable]} {
        if {[sizeof_collection \
            [dict get $xyz_s_drc_dict synchronizer_cells]] == 0} {
            display_message -error \
                "You can't enable XYZ_S3 without having specified the - \
                synchronizer_cells switch."
            set error_status 1
        }
        if {[get_dft_specification_requirement -logic_test] eq "off"} {
            set msg "You can't enable XYZ_S3 without also enabling logic_test \
            rules using the "
            append msg "\n'set_dft_specification_requirements -logic_test on' \
            command because it depends on "
            append msg "\nthe 'clock_domain_label' attribute set by DFT_C6."
            display_message -error $msg
            set error_status 1
        }
        if {$error_status} {
            return -code error
        }
    }
}
}
```

Example 4

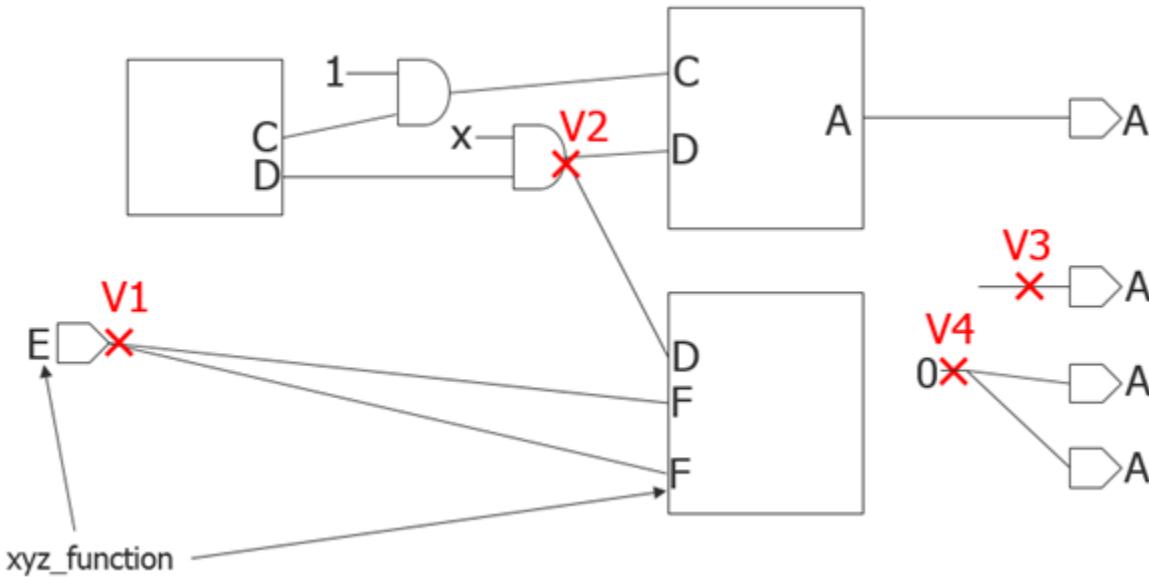
This example implements a DRC rule that verifies that output ports or input pins with a given xyz_function attribute value are controlled by input ports or output pins with the same xyz_function attribute value. This DRC could be used to verify that, for example, special pins of pad buffers are connected to the right programming logic. The kind of violation detected by this DRC is illustrated in [Figure 5-10](#). The names shown on the pins or ports are not their names but the value of the xyz_function attribute. The V1 violation detects an input port with xyz_function

equal to E driving two pins with xyz_function equal to F. The V2 violation detects two pins with xyz_function equal to D driven by a gate that blocks the controlling path from the valid source. The V3 violation is an output port with no source. Finally, the V4 violation is two output ports driven by a common source which is a constant value. The DRC is implemented such that it only issues one violation per invalid source and not one violation per incorrect destinations. You will see this is achieved by tracing backward from the output port and the input pins with an xyz_function value. When the controlling walk stops, a forward walk is used to collect all output ports and input pins with the same xyz_function value in the controlling fanout of the stop point. This avoids tracing backward from all incorrectly connected destination sharing a common source. This make sure that V1 and V2 in [Figure 5-10](#) are issued once instead of twice. For the V4 violation, because the first port to trace evaluate to a constant, a constant source tracing is done to find the source of the constant, the constant source is then forced to X and a forward controlling walk is performed to collect all destinations sharing the same invalid constant source.

Note

This DRC relies on having the user-defined attribute called xyz_function set on pins of instances and ports of the current design. Refer to Example 5 below for a description on how you can use configuration files to automate the process of setting the attribute.

Figure 5-10. Violations Detected by XYZ_S4_DRC



The xyz_attribute is registered in the file *XYZ_rule_set1_attribute_registrations.tcl* and its content was shown in [Example 3](#). Again the `trace_flat_model` command is used. This time the controllability switch is set to controlling and to constant when tracing a pin that has a constant value.

```
##### xyz_s4.tcl #####
proc XYZ_S4_drc {mode drc_name} {
    variable xyz_s_drc_dict
    if {$mode eq "enabled"} {
        return [dict get $xyz_s_drc_dict drc_indexes 4 enable]
    }
    # Stable_after_setup resets the ijttag network and include any
    # set_test_setup_icall that maybe needed to power-up the device
    # This will disable the DFT logic that may exists along the paths.
    set_current_simulation_context stable_after_setup

#Transfer the xyz_function from hierarchical model to flat model
#Used storage instead of inheritance because of performance and
#relatively low volume of data.

reset_attribute_value -object_type gate_pin -name xyz_function
#Using get_attributed_objects is instantaneous as compared to
#[get_pins -filter xyz_function]
foreach_in_collection attributed_object \
    [get_attributed_objects -attribute_name xyz_function \
        -object_types {port pin}] {
    set current_value [lindex [get_attribute_value_list \
        $attributed_object -name xyz_function] 0]
    set gate_pin [get_gate_pins -from_objects $attributed_object]
    set_attribute_value $gate_pin -name xyz_function \
        -value $current_value -silent
}

set gate_pins_with_xyz_function [get_attributed_objects \
    -attribute_name xyz_function -object_types gate_pin]
set destinations [filter_collection $gate_pins_with_xyz_function \
    "direction == input"]
while {[sizeof_collection $destinations] > 0} {
    set start_point [index_collection $destinations 0]
    set expected_value [lindex [get_attribute_value_list $start_point \
        -name xyz_function] 0]
    set simulation_value [lindex [get_attribute_value_list $start_point \
        -name simulation_value] 0]
    if {$simulation_value in {0 1}} {
        #Find source of constant to report it
        set stop_point [trace_flat_model \
            -from $start_point \
            -direction backward \
            -controllability constant \
            -store_path_to_stop_point path_to_stop_point \
            -map_tag_to_design_module_boundary on]

XYZ_S4_give_constantViolation $simulation_value $start_point \
    $stop_point $path_to_stop_point $expected_value
XYZ_S4_remove_all_destinations_with_that_common_source \
    $simulation_value $start_point $stop_point \
    $expected_value
    } else {
        set stop_point [trace_flat_model \
            -from $start_point \
            -direction backward \
            -stop_condition "xyz_function" \
            -controllability controlling \

```

```

        -store_path_to_stop_point path_to_stop_point \
        -map_tag_to_design_module_boundary on]

if {[sizeof_collection $stop_point] == 0} {
    XYZ_S4_give_no_sourceViolation $start_point $expected_value
    set destinations [remove_from_collection $destinations \
                      $start_point]
} else {
    set current_value [lindex [get_attribute_value_list $stop_point \
                                -name xyz_function] 0]
    if {$current_value ne $expected_value} {
        XYZ_S4_give_incorrect_sourceViolation $start_point \
            $stop_point $path_to_stop_point $current_value \
            $expected_value
    }
    XYZ_S4_remove_all_destinations_with_that_common_source \
        $simulation_value $start_point $stop_point \
        $expected_value
}
}

#####
proc XYZ_S4_give_no_sourceViolation {start_point expected_value} {
    set violation [DrcViolation #auto]
    set start_object_type [get_attribute_value_list [get_design_objects \
                                                    -from_objects $start_point] -name object_type]
    set start_object_name [get_single_name $start_point]
    $violation dict_set simulation_context [get_current_simulation_context]
    $violation dict_set list_of_paths side0 [list $start_point]
    $violation dict_set path_direction side0 backward
    $violation dict_set start_callout_string side0 \
        "Object with xyz_function=$expected_value with no source"
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' has no source."
    $violation dict_set message_string_list $msg_list
    add_drcViolation [namespace current]::$violation
}

#####
proc XYZ_S4_give_constantViolation {simulation_value start_point \
                                     stop_point path_to_stop_point expected_value} {
    set violation [DrcViolation #auto]
    set start_object_type [get_attribute_value_list [get_design_objects \
                                                    -from_objects $start_point] -name object_type]
    set start_object_name [get_single_name $start_point]
    set stop_object_type [get_attribute_value_list [get_design_objects \
                                                    -from_objects $stop_point] -name object_type]
    set stop_object_name [get_single_name $stop_point]
    $violation dict_set simulation_context [get_current_simulation_context]
    $violation dict_set list_of_paths side0 [list $path_to_stop_point]
    $violation dict_set path_direction side0 backward
    $violation dict_set start_callout_string side0 \
        "Start point with xyz_function=$expected_value"
    $violation dict_set end_callout_string side0 "Source of constant value"
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' is source by a constant $simulation_value."
    lappend msg_list "The source of the constant value is $stop_object_type
'$stop_object_name'."
```

```
$violation dict_set message_string_list $msg_list
    add_drcViolation [namespace current]::$violation
}
#####
proc XYZ_S4_give_incorrect_sourceViolation {start_point stop_point
path_to_stop_point current_value expected_value} {
    set start_port [index_collection $path_to_stop_point 0]
    set violation [DrcViolation #auto]
    set start_object_type [get_attribute_value_list [get_design_objects \
        -from_objects $start_point] -name object_type]
    set start_object_name [get_single_name $start_point]
    set stop_object_type [get_attribute_value_list [get_design_objects \
        -from_objects $stop_point] -name object_type]
    set stop_object_name [get_single_name $stop_point]
$violation dict_set simulation_context [get_current_simulation_context]
$violation dict_set list_of_paths side0 [list $path_to_stop_point]
$violation dict_set path_direction side0 backward
$violation dict_set start_callout_string side0 \
    "Start point with xyz_function=$expected_value"
if {$current_value eq ""} {
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' backward tracing stopped at"
    lappend msg_list "$stop_object_type '$stop_object_name' and this
$stop_object_type does not have an xyz_function value."
    $violation dict_set end_callout_string side0 \
        "Source with no xyz_function"
} else {
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' is sourced by"
    lappend msg_list "$stop_object_type '$stop_object_name' but its
xyz_function value is '$current_value'."
    $violation dict_set end_callout_string side0 "Source with incorrect
xyz_function"
}
$violation dict_set message_string_list $msg_list
    add_drcViolation [namespace current]::$violation
}
#####
proc XYZ_S4_remove_all_destinations_with_that_common_source
{simulation_value start_point stop_point expected_value} {
    upvar destinations destinations
    #This proc finds all the destination with the same xyz_funtion value
    #in the controlling fanout of the source to only give one violation
    #per bad source. It is also much more efficient when tracing a high
    #fanout function.
    set current_simulation_context [get_current_simulation_context]
    if {$simulation_value in {0 1}} {
        #If the stop point is constant, clone the simulation condition and\
        #force it to X to allow doing a controlling fanout trace.
        if {"xyz_delta" ni [get_simulation_context_list -user_defined]} {
            add_simulation_context xyz_delta -silent
        } else {
            set_current_simulation_context xyz_delta
        }
        copy_simulation_context -from $current_simulation_context
        add_simulation_forces $stop_point -values X
        simulate_forces
    }
}
```

```

set hits [trace_flat_model \
           -from $stop_point \
           -direction forward \
           -controllability controlling \
           -stop_condition "xyz_function" \
           -tag_condition "xyz_function==$expected_value"]
# In case the constant destination is not in controlling fanout when
# source is set to X
set destinations [remove_from_collection $destinations \
                  [add_to_collection $hits $start_point]]
if {[get_current_simulation_context] ne $current_simulation_context} {
    set_current_simulation_context $current_simulation_context
    # clean up memory
    delete_simulation_context xyz_delta
}
}

```

Example 5

This example does not show a new DRC example, but instead a method to populate the xyz_function attribute needed for the XYZ_S4 DRC rule shown in the previous example. Configuration data is a very convenient way of defining attributes on ports of a module. With a simple callback shown below, you can easily transfer the attribute values from the configuration file to the ports of the current design and the pins of the instances of the modules.

Below is an example configuration file describing attributes on ports of modules. Even though the example has multiple Core wrappers in the same file, you can choose to use one file per Core wrapper or merge them in a single file.

```

##### modules.tcd_xyz #####
Core(epoch_counter) {
    Xyz {
        Port(tic_enable) {
            function : tic_enable;
        }
    }
}
Core(code_gen) {
    Xyz {
        Port(tic_enable) {
            function : tic_enable;
        }
    }
}
Core(time_base) {
    Xyz {
        Port(tic_enable) {
            function : tic_enable;
        }
        Port(rstn) {
            function : reset;
        }
    }
}

```

```
Core(gps_baseband) {
    Xyz {
        Port(hw_rstn) {
            function : reset;
        }
    }
}
```

Below is the metadata used to define the legal syntax of the XYZ wrapper. See the “[Metadata Configuration Syntax](#)” chapter for more information about the metadata syntax. The location of the metadata file within the plugin was shown in [Figure 5-4](#) on page 1321.

```
##### XYZ_rule_set1.tessent_meta #####
Partitions: default, tcd;
Wrapper(/Core) {
    Wrapper(Xyz) {
        Repeatable: off;
        Property(version) {
            Value(version) {
                Type: int;
                DefaultValue : 1;
            }
        }
        Wrapper(Port) {
            Id(port_name) {
                Type: string;
            }
            Property(function) {
                Value(enum) {
                    LegalString : "",tic_enable, pre_tic_enable, reset;
                    DefaultValue : "";
                }
            }
        }
    }
}
```

You use the [read_core_descriptions](#) command to load the *.tcd_xyz* files. Below is a callback procedure that you can use in your plugin to automatically transfer the *xyz_function* value to the ports of the current design and pins of instances of the sub modules. Notice the use of the *-use_module_matching_options* switch in the [get_modules](#) command usage. This ensures that if your module names were unqualified during synthesis that the modules will still be matched to the RTL module names found in the Core wrapper. See the [set_module_matching_options](#) command for more information about module matching.

```
# Callback registration
namespace eval XYZ_rule_set1 {
    module_matching.post [create_wrapper_proc apply_xyz_function]
}
```

```

# Body of callback

proc apply_xyz_function {args} {
    reset_attribute_value -object_types {port pin} -name xyz_function
    foreach_in_collection tcd_xyz_wrapper [get_config_elements core/xyz \
                                              -partition tcd -silent] {
        set core_wrapper [get_config_value $tcd_xyz_wrapper -parent_object]
        set core_name [get_config_value $core_wrapper -id <0>]
        set matched_modules [get_module [list $core_name] \
                                     -use_module_matching_options -silent]
        if {[sizeof_collection $matched_modules] > 0} {
            if {[compare_collections $matched_modules \
                                         [get_current_design]] == 0} {
                set is_current_design 1
            } else {
                set is_current_design 0
                set matched_instances [get_instances \
                                         -of_modules $matched_modules -silent]
            }
            if {$is_current_design || \
                 [sizeof_collection $matched_instances] > 0} {
                foreach_in_collection port_wrapper [get_config_elements Port \
                                              -below_wrappers $tcd_xyz_wrapper -silent] {
                    set port_name [get_config_value $port_wrapper -id <0>]
                    set function_name [get_config_value function \
                                      -in_wrapper $port_wrapper]
                    if {$is_current_design} {
                        set objects [get_ports [list $port_name]]
                    } else {
                        set objects [get_pins [list $port_name] \
                                         -of_instances $matched_instances]
                    }
                    set_attribute_value $objects -name xyz_function \
                                       -value $function_name
                }
            }
        }
    }
}

```

register_drc_class

Context: dft. patterns, unspecified

Mode:setup

Registers a new class of DRC rules such that a DRC of that class can be registered

Usage

```
register_drc_class class_name [-runs_after class_name]
```

Description

A command used to register a new class of DRCs. Once you register a class, you use the “[register_drc](#) *drc_name -class class*” command to register DRC rules of that class. If you are registering your own DRC rules, it is best to use your own class name containing something specific to your company in the name such that you will never collide with future built-in DRC rules. In the example shown in the [register_drc](#) command section, the class name “XYZ_S” was used. The “XYZ” part is meant to represent your company and the “S” represent the further classification you may want to use.

Arguments

- ***class_name***

A required string that must start with a capital letter and is followed by only capital letters, numbers, and underscores.

- **-runs_after *class_name***

An optional switch-value pair that specifies the order in which the current DRC class is to run relative to the other custom-registered DRC classes. When unspecified, the class will run after the class that it follows in a numerical sorted list of the custom-registered class names.

When you register a DRC using the [register_drc](#) command, you can control precisely in which order the DRC runs. It is not a requirement that all DRC rules of a class run before the DRC rules of another class. You can intermix them as you needed. The class run order only affects registered DRC rules that do not use the -runs_after switch.

Examples

The following example registers a new DRC class called XYZ_T and specifies that it runs before the DRC class XYZ_D. If the -runs_after switch was not specified, the order would be reversed because XYZ_T is after the XYZ_D in a numerical sort.

```
register_drc_class XYZ_T -runs_after XYZ_D
```

register_static_dft_signal_names

Context: unspecified, all contexts

Mode: all modes

Registers new static DFT signal names and their attributes.

Usage

```
register_static_dft_signal_names {name ...}
    [-usage {global_dft_control | logic_test_control | scan_mode}]
    [-scan_mode_type {unwrapped | internal | external | retargeting}]
    [-default_value_in_all_test {auto | 0 | 1}]
    [-reset_value {0 | 1}]
    [-value_in_pre_scan_drc {x | 0 | 1}]
```

Description

Registers new static DFT signal names and their attributes. The tool has built-in pre-registered DFT signal names that you can view using the [report_dft_signal_names](#) command.

You use the register_static_dft_signal_names command to create a new dft_signal and specify its behavior (for example, reset_value, default_value_in_all_test).

You only use this command if the pre-defined dft_signals do not fit your needs. For example, if you have logic you want controlled in different test modes and none of the pre-defined dft_signals will work for your logic, then you can register a new dft_signal.

After you have registered your dft_signal, you use the [add_dft_signals](#) command to create the hardware needed to implement your new dft_signal. You are now able to use your new dft_signal to control logic by referencing your new dft_signal with the “dft_signal_source_name” option of the [add_dft_control_points](#) or [add_dft_clock_mux](#) commands.

You can also register new scan_mode signals using this command if you find that you do not have the one you need in the pre-registered list. It is important that you set the proper -scan_mode_type value when registering a DFT signal of usage scan_mode. See the description of the [set_static_dft_signal_values](#) command to understand how the scan_mode_type affects the implied value of the int_ltest_en and ext_ltest_en signals.

Arguments

- ***name***

A required repeatable string that specifies the name of the DFT signals to register. The string is also allowed to be a well formatted Tcl list of names. The names must start with a letter and be followed by only letters, numbers, and/or underscores.

- **-usage {global_dft_control | logic_test_control | scan_mode}**

An optional switch and literal pair that specifies the usage of the newly-registered DFT signal names. When the -usage switch is not specified, the default is global_dft_control.

Choose one from the following:

global_dft_control — This usage is for signals controlling global resources like clocking and power management signals. Those signals are automatically set to their specified “default_value_in_all_test” values unless the value is explicitly requested to be differently set using either the DftControlSetting wrapper of the [Patterns](#) wrapper or using the [set_static_dft_signal_values](#) command. See [Table 3-10](#) on page 190 in the [add_dft_signals](#) command description of the pre-registered global DFT controls.

logic_test_control — This usage is for signals used to control aspects of the logic test modes. Unlike the signal with usage scan_mode, the signals are not mutually exclusive to each other and can be set high or low when using the same scan configuration. See [Table 3-10](#) on page 190 in the [add_dft_signals](#) command description of the pre-registered logic test controls.

scan_mode — This usage is for signals used to control the scan chain configurations within a module. All scan_mode signals are mutually exclusive and only one of them per instance can be set to 1.

- **-scan_mode_type {unwrapped | internal | external | retargeting}**

An optional switch and literal pair that specifies the type of the scan_mode signal. You must ensure you set the proper -scan_mode_type value when registering a DFT signal of usage scan_mode. See the description of the [set_static_dft_signal_values](#) command to understand how the scan_mode_type affects the implied value of the int_ltest_en and ext_ltest_en signals.

When the -scan_mode_type switch is not specified, the default is unwrapped.

Choose one from the following:

unwrapped — Used for scan_mode signal enabling scan configuration having no wrapper isolation. When a scan_mode signal having the type unwrapped is set to 1 using the [set_static_dft_signal_values](#) command, the int_ltest_en and ext_ltest_en signal that may exist in the same instance are automatically set to 0.

internal — Used for scan_mode signal enabling scan configuration using the inward isolation of the wrapper chains. When a scan_mode signal having the type internal is set to 1 using the [set_static_dft_signal_values](#) command, the int_ltest_en and ext_ltest_en signal that typically exist in the same instance are automatically set to 1 and 0, respectively.

external — Used for scan_mode signal enabling scan configuration using the outward isolation of the wrapper chains. When a scan_mode signal having the type external is set to 1 using the [set_static_dft_signal_values](#) command, the int_ltest_en and ext_ltest_en signal that typically exist in the same instance are automatically set to 0 and 1, respectively.

retargeting — Used for scan_mode signal enabling of channel access to combinations of lower-level blocks. The lower-level blocks typically have one of their internal scan_mode signals activated.

- **-default_value_in_all_test {reset_value | 0 | 1}**

An optional switch and literal pair that specifies the default value the signals with usage global_dft_control should have in all tests. The default value of reset_value means that the value specified for the -reset_value switch is also used as the default value in all tests.

- **-reset_value {0 | 1}**

An optional switch and literal pair that specifies the reset value for the corresponding global_dft_control signal. This value corresponds to the value you want this signal to have in functional mode. It is illegal to specify -reset_value 1 when the -usage switch is not global_dft_control. If you want to register a DFT signal with a reset value of 1, then you must register the signal as global_dft_control.

- **-value_in_pre_scan_drc {x | 0 | 1}**

An optional switch and literal pair that specifies the value the logic_test_signal should be set to when running the Pre-DFT DRC rules. This value, with the Xs converted to 0s, also serves as the default value the given logic test control is set to when using the [set_static_dft_signal_values](#) command. See the [set_static_dft_signal_values](#) section for a description of where this value is used.

Examples

Example 1

The following example registers a new global DFT control signal to control the select of a clock multiplexer. By default, the select is driven high in all DFT test modes but the select can be set to 0 using the DftControlSettings wrapper in the [Patterns](#) wrapper or using the [set_static_dft_signal_values](#) command to configure the test_setup of a logic test mode.

```
register_static_dft_signal_name clockb_select -reset_value 0 -default_value_in_all_test 1
add_dft_control_points clock_mux/s -dft_signal_source_name clockb_select
```

Example 2

The following example illustrates how you can create a plugin to augment the *.tcd_scan* syntax with a new logic_test_control DFT signal. The DFT signal will automatically be registered. If included inside a *.tcd_scan* file, the DFT signal will automatically be understood during sub-chain tracing in context dft -scan (see [set_context](#)). Also, if the port is seen to be tied off on the instance, the scan insertion tool will automatically attach the port to the source of the DFT signal as returned by the [get_dft_signal](#) command.

Below is the content of file *path/my_plugin/metadata/tcd_scan.tessent_meta*. As you can see, it defines a new logic_test_control called my_sig1. The tracing_condition property defines the value that must be forced on the signal when tracing the sub-chain. This value becomes the “value_in_pre_scan_drc” value of the registered DFT signal. You must use a different usr[#]

index for each signal you registered. In the following example, you can only modify the items in bold. The rest of the entries must appear exactly as shown.

```
Partitions: default,tcd;

Wrapper (/Core/Scan) {
    Wrapper (MySig1) {
        Options(PreDftDrcs) {
            dft_control_type : scan_control;
            dft_scan_signal : usr[0]:0;
            dft_scan_function : my_sig1; //Any name
            tracing_condition : active; //active or inactive
            reset_state : inactive;//active or inactive
        }
        Id(port_name*) {
            type : string;
        }
        // use active_polarity_default_all_zeros when the default
        // active polarity of the signal is 0.
        reference =
    /tcd_scan/active_polarity_default_all_ones/active_polarity;
    }
}
```

To have the metadata automatically loaded by the tool when you invoke Tesson Shell, set the TESSENT_PLUGIN_PATH environment variable to *path/my_plugin*. For more information about using plugins, refer to “[Adding Custom Plugin Functionality](#)” on page 3921.

Related Topics

[report_static_dft_signal_settings](#)
[unregister_static_dft_signal_names](#)

register_tcl_command

Context: unspecified, all contexts

Mode: all modes

Registers a Tcl command and makes it available in the global name space.

Usage

```
register_tcl_command command_name
  {-syntax_bnf syntax_bnf}
  {-tcl_proc tcl_proc}
  {-context_list context_list}
  {-system_mode_list system_mode_list}
  [-resource_list resource_list]
  [-extra_help extra_help]
  [-integer_list_option integer_list_option]
  [-string_list_option string_list_option]
  [-real_list_option real_list_option]
  [-integer_list_value integer_list_value]
  [-string_list_value string_list_value]
  [-real_list_value real_list_value]
  [-boolean_option boolean_option]
  [-integer_option integer_option]
  [-string_option string_option]
  [-real_option real_option]
  [-integer_value integer_value]
  [-string_value string_value]
  [-real_value real_value]
  [-transcript_style {tool_command | normal}]
```

Description

Registers a Tcl command and makes it available in the global name space.

The registered command appears as a normal tool command with proper option parsing and online help messages. The body of the registered command is implemented in a Tcl proc referenced by the -tcl_proc option. After the options are validated, they are packaged into a Tcl list of option-value pairs that are packed into a Tcl array for easy random access as shown in [Example 1](#).

You control the availability of the newly registered command using the -context_list and -system_mode_list options.

Note

 Commands in Tesson Shell are not case-sensitive. Therefore, if you register a Tcl command that contains upper-case letters, the tool will not preserve the case of that command, and they will be displayed as lower case in the online help.

When registering a Tcl command with embedded commands, you use a double hyphen (--) to specify the end of the command options passed to the embedded command. Passing an argument value to the registered Tcl command that starts with a hyphen (-) can result in an error when the value is ultimately used as an argument to an embedded command in the proc that is registered.

The following Tesson Shell transcript shows the behavior of the [is_collection](#) command in various situations that require the usage of double hyphens:

```
SETUP> set v -a
-a
# Here, 'is_collection' wrongly tries to interpret '-a' as one of its
# arguments
SETUP> is_collection $v
// Error: Unrecognized argument '-a'. ( help is_collection )

# With the double hyphen, the value is recognized as NOT being a
# collection.
SETUP> is_collection -- $v
0
# Other arguments recognized by the command can be passed BEFORE the
# double hyphen
SETUP> is_collection -single_type_object -- $v
0
SETUP>
```

The above usage of the double hyphen (--) is demonstrated in the [Example 1](#).

Arguments

- ***command_name***

A required string that specifies the name of the command to register. Because the command will exist in the global name space, its name must be unique so that it does not collide with other already registered commands. You can unregister any registered Tcl command using [unregister_tcl_command](#) until you execute the [lock_current_registration](#) command. After you issue [lock_current_registration](#), all registered commands and attributes are locked and can no longer be modified.

All commands loaded with an automated plugin mechanism are locked before the interactive shell is released to the user. “[Example 2](#)” on page 1367 explains how to initially debug your commands interactively when it is still possible to unregister them, and then shows you how to package them into a plugin directory which is automatically loaded at tool invocation. For more information about plugins, see the [lock_current_registration](#) command.

- **-syntax_bnf syntax_bnf**

A required switch and string value pair that specifies the grammar rules of your command using simple Backus-Naur Form (BNF). You should place the syntax_bnf string in braces to avoid needing to escape square brackets which are used to express that an option or value is optional. You should use parentheses to group options that are exclusive to others. You can use the | symbol to denote exclusivity of options. BNF uses the symbols you defined when declaring the *option and *value arguments; the symbols must always be enclosed in <> characters.

The following example defines five arguments; the first two are required and the last three are optional. The fifth argument is mutually exclusive of the third and fourth arguments. An argument is repeatable if its tag is declared with an * as its last character.

```
{ <arg1> <arg2> ([arg3] [arg4]) | [arg5] }
```

- **-tcl_proc tcl_proc**

A required switch and string pair that specifies the name, including its namespace, of the Tcl proc this command will call. To facilitate packaging the new command as a plugin that is automatically loaded into local namespaces, you should specify tcl_proc using the following format: [namespace current]::proc_name. This format ensures that the registered Tcl command always knows where to look for the Tcl proc regardless of which namespace the command is evaluated in.

For easy interactive debugging that can be easily deployed as a plugin, you should package both your Tcl proc and the register_tcl_command into a single file and surround it with a namespace eval name_space {} wrapper. Do not put :: in front of the specified name_space string so that this namespace string will be automatically appended to the one automatically created for all plugins. See the examples for more information on how to do this.

- **-context_list context_list**

A required switch and string pair that specifies a Tcl list of contexts the command is enabled in. Each element of the Tcl list is a context with an optional sub-context option, as shown here:

```
-context_list {{dft} {patterns -ijtag}}
```

The available literals are: all, dft, patterns. The available sub-contexts are listed under “[Contexts](#)” in the *Tessent Shell User’s Manual*.

- **-system_mode_list system_mode_list**

A required switch and string pair that specifies a Tcl list of system modes in which you want to enable the command. The available literals are: all, setup, analysis, and insertion.

- **-resource_list resource_list**

An optional switch and string pair that specifies the list of resources required for the command to be available; if any of the specified resources is not available, the command is

not available. The resource_list argument is a Tcl list of strings that identify the resources required. Possible values are:

`design_is_elaborated` — Used to check that the hierarchical data model has been created using the `set_current_design` command.

`design_modules_exist` — Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the `get_modules` command.

`flat_model_exists` — Used to check that the flat data model is available. The flat data model was created by either the `create_flat_model` command or by the “`set_system_mode analysis`” command.

`ic1_is_elaborated` — Used to check that the ICL hierarchical data model exists. It exists after the `set_current_design` command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

`ic1_modules_exist` — Used to check that at least one ICL module exists in memory and can be accessed using the `get_icl_modules` command.

`rtl_mode_enabled` — Used to check that the `-rtl` option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

- **-extra_help extra_help**

An optional switch and string pair that enables you to define a multi-line help paragraph that is displayed at the bottom of the online help. You can use `\n` to break your strings into multiple lines. The easiest way to create a multi-line help string is to construct it line by line as shown here:

```
set extra_help "This is line1\n"
append extra_help "and this is line 2\n"
append extra_help "and this the last line."
register_tcl_command -extra_help "$extra_help" ...
```

- **-integer_list_option integer_list_option**

An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of integers. The `integer_list_option_list` string is a Tcl list with three required elements and five optional elements as shown here:

```
{ {<symbol>} {-option_name} {tag} [{description_string}] [{range}]
[ {enum_list} ] [{<help_symbol>}] [{<visibility>}]}]
```

`<symbol>` — The first element of the list is required. It defines a unique symbol for the argument that you use in the `-syntax_bnf` `syntax_bnf` string to refer to it and it also displays in the online help description. The symbol string must always begin with a `<` character and end with a `>` character.

`-option_name` — The second element of the list is required. It defines a unique option name. The option name must start with a `-` character. The online help displays this option as shown here: “`-option_name <symbol>`”.

`tag` — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an `*`

character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

```
{ {range,0}{value1}{range,1}{value2}{range,2}{value3} }
```

description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest allowed value is a MIN_INT. When the second integer is not specified, the largest allowed value is MAX_INT. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

enum_list — The sixth element is optional. It allows you to specify strings as allowed values. For example, you may want to define an option that allows a positive integer or the literal “last”. This is achieved by using a minimum value of 0 and an enum string called “last”.

<help_symbol> — The seventh element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eighth element.

visibility — The eighth element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- **-string_list_option** *string_list_option*

An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of strings. The string_list_option is a Tcl list with three mandatory elements and four optional elements as shown here:

```
{ {<symbol>} {-option_name} {tag} [{description_string}]  
[ {enum_list} ] [ {<help_symbol>} ] [ {<visibility>} ] }
```

<symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also

displays in the online help description. The symbol string must always begin with a < character and end with a > character.

-option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

```
{ {name, 0} {value1} {name, 1} {value2} {name, 2} {value3} }
```

description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

enum_list — The fifth element is optional. It allows you to specify strings as allowed values. When omitted or specified as an null string, the string_list_option allows any string values.

<help_symbol> — The sixth element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

visibility — The seventh element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- -real_list_option *real_list_option*

An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of real numbers. The real_list_option is a Tcl list with three required elements and five optional elements as shown here:

```
{ {<symbol>} {-option_name} {tag} [{description_string}] [{range}]  
[{enum_list}] [{<help_symbol>}] [{<visibility>}]}
```

<symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also

displays in the online help description. The symbol string must always begin with a < character and end with a > character.

-option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

```
{ {range,0}{value1}{range,1}{value2}{range,2}{value3} }
```

description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest allowed value is a MIN_DOUBLE. When the second integer is not specified, the largest allowed value is MAX_DOUBLE. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

enum_list — The sixth element is optional. It allows you to specify strings as allowed values. For example, you may want to define an option that allows a positive integer or the literal “last”. This is achieved by using a minimum value of 0 and an enum string called “last”.

<help_symbol> — The seventh element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[<parent_of_instances> | <instance_objects> | <below_instances> | <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eighth element.

visibility — The eighth element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- **-boolean_option boolean_option**

An optional switch and string pair that defines an argument that is an option without a value. The boolean_option string is a Tcl list with three required elements and two optional elements as shown here:

```
{ {<symbol>} {-option_name} {tag} [{description_string}]  
[ {<visibility>} ] }
```

<symbol> — The first element of the list is required and defines a unique symbol for the argument. You use this element in the -syntax_bnf syntax_bnf string to refer to it. The symbol string must always begin with a < character and end with a > character.

-option_name — The second element of the list is required and defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name”.

tag — The third element of the list is required and defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc.

description_string — The forth element is optional and can be used to define a one-line description for the argument that displays in the online help.

visibility — The fifth element is optional and can be used to specify whether the option is included in the help message. You specify a null string “” if you want the option to show up in the help message and you specify the string “hidden” if you want the option to be hidden.

- **-integer_list_value integer_list_value**

An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

The integer_list_value string is a Tcl list with two required elements and five optional elements. This string has exactly the same definition as the integer_list_option string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with an * character because only options can be defined as repeatable.

```
{ {<symbol>} {tag} [{description_string}] [{range}] [{enum_list}] [{<help_symbol>}]  
[ {<visibility>} ] }
```

- **-string_list_value string_list_value**

An optional switch and string pair that is identical to the -string_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

The string_list_value string is a Tcl list with two required elements and four optional elements. This string has exactly the same meaning as the string_list_option string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with an * character as only options can be defined as repeatable.

```
{ {<symbol>} {tag} [{description_string}] [{enum_list}] [{<help_symbol>}]  
[ {<visibility>} ] }
```

- **-real_list_value *real_list_value***

An optional switch and string pair that is identical to the -real_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

The *real_list_value* string is a Tcl list with two required elements and five optional elements. This string has exactly the same meaning as the *real_list_option* string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with a * as only options can be defined as repeatable.

```
{ {<symbol>} {tag} [{description_string}] [{range}] [{enum_list}] [{<help_symbol>}]  
[ {<visibility>} ] }
```

- **-integer_option *integer_option***

An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that only a single integer is allowed as the value as opposed to a Tcl list of integers.

- **-string_option *string_option***

An optional switch and string pair that is identical to the -string_list_option switch and string pair except that only a single string is allowed as the value as opposed to a Tcl list of strings.

- **-real_option *real_option***

An optional switch and string pair that is identical to the -real_list_option switch and string pair except that only a single real is allowed as the value as opposed to a Tcl list of real numbers.

- **-integer_value *integer_value***

An optional switch and string pair that is identical to the -integer_list_value switch and string pair except that only a single integer is allowed as the value as opposed to a Tcl list of integers.

- **-string_value *string_value***

An optional switch and string pair that is identical to the -string_list_value switch and string pair except that only a single string is allowed as the value as opposed to a Tcl list of strings.

- **-real_value *real_value***

An optional switch and string pair that is identical to the -real_list_value switch and string pair except that only a single real number is allowed as the value as opposed to a Tcl list of real numbers.

- **-transcript_style {tool_command | normal}**

An optional switch and literal that specifies whether the command transcripts like a normal command or like a tool command. The default is *tool_command*.

tool_command —Transcripts tool commands, including all arguments, as the commands execute. The command string is transcribed after being evaluated by Tcl so all

variable and command references are resolved. For more information on how a tool command transcripts, see the “[set_transcript_style full](#)” command and option.
normal — Transcripts the command before any Tcl evaluation is done. This includes all Tcl commands and constructs. For more information on how a tool command transcripts, see the “[set_transcript_style tool_commands_only](#)” command and option.

Examples

Example 1

The following example defines a simple proc called “example” that displays the arguments it receives. The register_tcl_command defines a command called “example” that accepts a repeatable integer list option, a non-repeatable string list option, and a string value. Notice that the registered command will be located in the global namespace and the Tcl proc that it calls is located in the ::ExampleCommands namespace so that the two identical proc names do not collide.

```
namespace eval ExampleCommands {
    proc example {args} {
        # Initialize default values
        array set all_args {object ""}
        # convert args list into an array for easy random access
        array set all_args $args
        puts "chain = $all_args(chain)"
        # Without '--', 'is_collection' would fail if 'all_args(object)'
        # is a string that starts with a '-' since it would try to interpret
        # this value as an option to the command.
        if { [is_collection -single_type_objects -- $all_args(object)] && \
            [sizeof_collection $all_args(object)] > 0 } {
            puts "Object name(s) { [lget_name_list -silent $all_args(object)] }"
            puts -none newline " of type "
            set obj [index_collection $all_args(object) 0]
            puts "'[get_attribute_value_list -name object_type $obj -silent]'"
        } elseif { [is_collection -- $all_args(object)] && \
            [sizeof_collection $all_args(object)] > 0 } {
            foreach_in_collection obj $all_args(object) {
                puts -none newline "Object name '[get_single_name $obj -silent]' \
                    of type "
                puts "'[get_attribute_value_list -name object_type $obj -silent]'"
            }
        } else {
            puts "Object name(s) verbatim '$all_args(object)'"
        }
        # Range is repeatable, so its tags are indexed
        for {set i 0} {1} {incr i} {
            if {[info exists all_args(range,$i)]} {
                puts "range $i = $all_args(range,$i)"
            } else {
                break
            }
        }
    }
}
```

```
register_tcl_command example \
-string_value {{<chain_name>} {chain}} \
-string_list_option { {<object>} {-object} {object} } \
-integer_list_option { {<range>} {-range} {range*} } \
-syntax_bnf {<chain_name> [<object>] [<range>]} \
-context_list {dft patterns} \
-system_mode_list {setup insertion} \
-tcl_proc "[namespace current]::example"
}
```

Here is a transcript of a Tessent Shell session that uses that register command, assuming that the above content has been sourced beforehand:

```
SETUP> set_context dft -rtl
# show the help
SETUP> example
// Error: Missing options ( <chain_name> [ -object <object> ] 
// [ {-range <range> ...} ] ) ( help example )

# Basic usage with single collection type for '-object'
SETUP> example my_chain
chain = my_chain
Object name(s) verbatim ''

SETUP> set C [add_config_element tmp(1)]
// sub-command: add_config_element tmp(1)
{/tmp(1)}

SETUP> example my_chain -object $C
chain = my_chain
Object name(s) { /tmp(1) }
of type 'config_wrapper'

SETUP> append_to_collection C [add_config_element tmp(2)]
{/tmp(1) /tmp(2)}

SETUP> example my_chain -object $C
chain = my_chain
Object name(s) { /tmp(1) /tmp(2) }
of type 'config_wrapper'

# Example with mixed collection

SETUP> set_system_mode insertion
INSERTION> set m [create_module my_mod]

// sub-command: create_module my_mod
{my_mod}
```

INSERTION> append_to_collection C \$m

{ /tmp(1) /tmp(2) my_mod}

INSERTION> example my_chain -object \$C

```
chain = my_chain
Object name '/tmp(1)' of type 'config_wrapper'
Object name '/tmp(2)' of type 'config_wrapper'
Object name 'my_mod' of type 'module'

# Show what happens when passing lists instead of objects
```

Empty list

INSERTION> example my_chain -object [list]

```
chain = my_chain
Object name(s) verbatim ''

# List which starts with a '-'
```

INSERTION> example my_chain -object [list -a]

```
chain = my_chain
Object name(s) verbatim '-a'

# List with many elements
```

INSERTION> example my_chain -object [list -a -b -c]

```
chain = my_chain
Object name(s) verbatim '-a -b -c'

# Show how the '-range' can be repeated
```

INSERTION> example my_chain -range 1 -range 4 -range 10

```
chain = my_chain
Object name(s) verbatim ''
range 0 = 1
range 1 = 4
range 2 = 10
```

INSERTION>

Example 2

The following example defines a wrapper Tcl proc that is used to call the intercept_with_cell procedure described in [Example 5](#) of the get_dft_cell command. The code is surrounded in a namespace eval block which puts the intercept_with_cell_wrapper proc in that name space and avoids polluting the global namespace with the Tcl procs when sourcing it. The [namespace current] command, used with the register_tcl_command -tcl_proc option, ensures the tool knows to search in the correct name space. It also ensures that the registered Tcl command in the global namespace does not collide with the Tcl proc of the same name.

After you verify that your registered commands and/or attributes work correctly, you can make them into a *plugin* directory that Tessent Shell automatically loads during initialization. To create the plugin directory, simply place your files in a directory called <plugin>/tcl_modules/<MySpace> where <plugin> and <MySpace> are arbitrary directory names. Set a TESSENT_PLUGIN_PATH environment variable to point to the <plugin> directory and any files with a .tcl extension found inside <plugin>/tcl_modules/<MySpace> will be evaluated into the Tcl namespace ::tcl_modules::<MySpace>. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon. A utility command initialize_module is used to load all files ending with .tcl found inside a local directory having the exact same name as the name space.

In the example below, the namespace is called InterceptWithCell which means that initialize_module will source all files matching *./InterceptWithCell/*.tcl*. The advantage of using the initialize_command is that the Tcl proc is only loaded the first time it is needed.

If you name your <plugin> directory name “tessent_plugin” and place it one level above the bin directory of Tessent Shell, the directory will be automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the [lock_current_registration](#) command is issued so that they cannot be unregistered accidentally.

```
namespace eval InterceptWithCell {  
    proc intercept_with_cell_wrapper {args} {  
        initialize_module  
        intercept_with_cell_body {*}$args  
    }  
    set extra_help "Command accepts a pin, port, or net object as first"  
    append extra_help "argument. It is intercepted with cell returned by \n"  
    append extra_help "get_dft_cell <cell_function_name>."  
  
    set cell_list "One of mux, clock_mux, and, clock_and, clock_or,"  
    append cell_list "buffer, clock_buffer, or inverter."  
  
    set bnf {<node> <function_name> [<input2_source>] [<select_source>] }  
    append bnf {[<level>] [<reset_intercept_list>]}  
}
```

```
register_tcl_command intercept_with_cell \
    -context_list {dft} \
    -system_mode_list {insertion} \
    -resource_list {design_is_elaborated} \
    -extra_help $extra_help \
    -string_value {"<node>" {node} {} } \
    -string_option [list "<function_name>" "-cell_function_name" \
        {cell_sel} $cell_list {mux clock_mux and clock_and or \
        clock_or buffer clock_buffer inverter}] \
    -string_option {"<input2_source>" "-input2" {input2} {} } \
    -string_option {"<select_source>" "-select" {select} {} } \
    -integer_option {"<level>" "-cascading_level" {level} {} {1 9}} \
    -boolean_option {"<reset_intercept_list>" "-reset_intercept_list" \
        {clear}} \
    -syntax_bnf "$bnf" \
    -transcript tool_command \
    -tcl_proc "[namespace current]::intercept_with_cell_wrapper"
}
```

The online help for the registered command is shown here:

```
INSERTION> help intercept_with_cell
intercept_with_cell <node> -cell_function_name <function_name>
    [-input2 <input2_source>] [-select <select_source>]
    [-cascading_level <level>] [-reset_intercept_list]
```

DESCRIPTION

-cell_function_name: One of mux, clock_mux, and, clock_and, or,
clock_or, buffer, clock_buffer

Command accepts a pin, port, or net object as first argument. It is
intercepted with cell returned by get_dft_cell <cell_function_name>.

Related Topics

[display_message](#)

[lock_current_registration](#)

[unregister_tcl_command](#)

[remove_from_collection](#)

Context: unspecified, all contexts

Mode: all modes

Removes objects found in obj_spec from a collection, resulting in a new collection.

Usage

`remove_from_collection base_collection obj_spec`

Description

Removes objects found in obj_spec from a collection, resulting in a new collection.

The base_collection remains unchanged.

If obj_spec is a null collection, the resulting collection is a copy of the base collection. If everything in base_collection is included in obj_spec, the result is an empty collection.

Arguments

- ***base_collection***

A required value that specifies the base collection from which objects within obj_spec are to be removed.

- ***obj_spec***

A required value that specifies a Tcl list of one or more object names, or a collection of one or more objects.

Examples

The following example creates a collection of all input and inout ports minus the port called CLK.

```
set data_in [remove_from_collection [get_ports -direction input inout] CLK]
{"in1", "in2"}
```

Related Topics

[add_to_collection](#)

[append_to_collection](#)

[compare_collections](#)

[copy_collection](#)

[index_collection](#)

[is_collection](#)

[filter_collection](#)

[foreach_in_collection](#)
[range_collection](#)
[sizeof_collection](#)
[sort_collection](#)

rename_instance

Context: all contexts

Mode: insertion

Renames the leaf name of an instance.

Usage

```
rename_instance obj_spec -new_leaf_name leaf_inst_name [-silent]
```

Description

Renames the leaf name of an instance.

This command returns a collection containing the renamed instance.

If the instance within *obj_spec* does not exist or if an object already exists with the same *leaf_inst_name* within the *parent_instance*, the tool generates an error. You can use the *-silent* switch to specify to not generate an error message and ignore the renaming request in which case the command returns a null collection.

Arguments

- *obj_spec*

A required value that specifies a single instance name or a collection of a single instance object as returned by the [get_instances](#) command.

- **-new_leaf_name *leaf_inst_name***

A required switch and value pair that specifies the new leaf name to assign to the instance object referenced in *obj_spec*.

- *-silent*

An optional argument that specifies to ignore any of the two following error conditions: the instance within *obj_spec* does not exist or if an object already exists with the same *leaf_inst_name* within the *parent_instance*.

Examples

Example 1

The following example renames the leaf name and3 to my_and3.

```
rename_instance /modB/and3 -new_leaf_name my_and3
```

Example 2

The following example identifies all instances having the user-defined attribute *is_my_type* set to true and adds a special prefix called “myPrefix_” to their leaf instance name such that it will be easier to find them in the downstream steps of the flow using the SDC command “*get_cells myPrefix_* -hierarchical*”.

```
foreach_in_collection inst [ get_instance -filter "is_my_type"] {  
    set leaf_name [get_attribute_value_list $inst -name leaf_name]  
    rename_instance $inst -new_leaf_name myPrefix_${leaf_name}  
}
```

Related Topics

[create_instance](#)
[delete_instances](#)
[get_common_parent_instance](#)
[replace_instances](#)
[uniquify_instances](#)

rename_net

Context: dft

Mode: insertion

Renames the leaf name of a net.

Usage

```
rename_net obj_spec -new_leaf_name leaf_net_name [-silent]
```

Description

Renames the leaf name of a net.

If a net already exists with the same *leaf_net_name*, the tool chooses an unused net name by adding the net unqualification suffix.

Arguments

- ***obj_spec***
A required value that specifies a single net name or a collection of a single net objects as returned by the [get_nets](#) command.
- **-new_leaf_name *leaf_net_name***
A required switch and value pair that specifies the new leaf name to assign to the net object referenced in *obj_spec*.
- **-silent**
An optional argument that specifies to ignore the following error condition: the net within *obj_spec* does not exist.

Examples

Example 1

The following example demonstrates the use of this command:

```
INSERTION> get_nets
{joe[1] joe[0]}

INSERTION> rename_net [get_nets joe] -new_leaf_name jack
{jack[1] jack[0]}

INSERTION> get_nets
{jack[1] jack[0]}
```

Example 2

The following example illustrates the uniquification suffix if a net already exists with the same *leaf_net_name*:

```
INSERTION> get_nets
{w1 w2}

INSERTION> rename_net [get_net w1] -new_leaf_name w2
{w2_ts1}
```

replace_instances

Context: dft

Mode: insertion

Replaces the module definition of an instance with a new module definition.

Usage

```
replace_instances obj_spec1 -with_module obj_spec2 [-silent]
```

Description

Replaces the module definition of an instance with a new module definition. The module definition assigned to the instance during instantiation is replaced with the module specified by the module within *obj_spec2*. If the new module has new ports that have the [function](#) attribute set to “tie0” or “tie1”, then these are automatically tied on the new instance pins.

Connections to pins that exist on both the original and new module remain unchanged. New pins that did not exist on the original module are left open after the replacement. Pins that existed on the original module but do not exist on the new module are silently disconnected.

This command returns a collection containing the new instance.

If *obj_spec1* does not exist, the tool generates an error. You can use the -silent switch to specify to not generate an error message when the object within *obj_spec1* does not exist. If *obj_spec2* is the same module as the original module, no change is made.

Arguments

- ***obj_spec1***

A required value that is Tcl list of one or more instance names, or a collection of one or more instances as generated by the [get_instances](#) command.

- **-with_module *obj_spec2***

A required switch and value pair that specifies the new module object to assign to the instances within *obj_spec1*. The *obj_spec2* value must be a single module name or a collection of a single module object as created by the [s](#) command.

- **-silent**

An optional argument that specifies to ignore specified objects of *obj_spec1* or *obj_spec2* that do not exist and to not generate error messages.

Examples

Example 1

The following example replaces the module definition of the instance u1/u2 with the module named nor2.

```
replace_instances u1/u2 -with_module nor2
```

Example 2

The following example replaces the module definition for instance u1/u2 to ModB. The original connection to pin u1/u2/a is preserved while the new pin u1/u2/b is left open.

```
get_attribute_value_list u1/u2 -name module_name
```

```
{ModA}
```

```
get_fanin u1/u2/a -stop_on net
```

```
{u1/n45}
```

```
replace_instances u1/u2 -with_module ModB
```

```
{u1/u2}
```

```
get_fanin u1/u2/a -stop_on net
```

```
{u1/n45}
```

```
get_fanin u1/u2/b -stop_on net
```

```
{}
```

Related Topics

[create_instance](#)

[copy_module](#)

[delete_instances](#)

[get_common_parent_instance](#)

[uniquify_instances](#)

rename_module

Context: dft

Mode: insertion

Assigns a new name to an existing module.

Usage

```
rename_module object_spec -new_name module_name [-silent]
```

Description

Assigns a new name to an existing module. If the *object_spec* is a parameterized view of a module, then only the particular view is renamed.

When specified, all instances of *object_spec* in the hierarchy will be bound to the new module. The old module still exists and instances that are not in the current hierarchy are not bound to the new module, but are still bound to the old one.

Arguments

- ***object_spec***

A required string or collection that specifies the module.

The *object_spec* must match exactly one module/parameterized view otherwise an error is given.

- ***-new_name module_name***

A required switch and string that specifies the new module name for *object_spec*.

- ***-silent***

An optional argument that suppresses the error issued if the *object_spec* does not exist or if more than one module is specified.

Related Topics

[rename_instance](#)

report_aborted_faults

Context: dft -edt, patterns -scan

Mode: analysis

Displays information on some potentially testable faults that remain undetected (UD) after the ATPG process.

Usage

`report_aborted_faults [format_type] [>|>>] file_pathname]`

Description

Displays information on testable faults that remain undetected (UD) after the ATPG process.

The report_aborted_faults command can help you determine why the tool aborted faults in the undetected fault list. You can then analyze whether to change the current abort limit to possibly allow ATPG to generate tests for those faults before the tool aborts them. To change the abort limit, use the [set_abort_limit](#) command.

Arguments

- *format_type*

An optional literal that specifies the type of information that you want the tool to display regarding the aborted faults. The literal choices for the *format_type* argument are as follows:

SUMMARY — A literal that displays a summary of the number of aborted faults for each category. This is the default.

All — A literal that displays all the aborted faults that are currently in the undetected fault list.

Backtrack — A literal that displays all the aborted, undetected faults that exceeded the backtrack limit.

CLOCK_control — A literal that displays all the aborted, undetected faults that are due to clock control.

CONtention — A literal that displays all the aborted, undetected faults due to bus contention or ram write contention.

DECisions — A literal that displays all the aborted, undetected faults that exceeded the maximum number of decisions.

DETected — A literal that displays all the faults that the tool aborted and then later detected.

Edt — (**Available only when EDT is on**) A literal that displays EDT aborted faults.

When the tool generates an effective fault test but is unable to compress the pattern, the fault is classified as an EDT aborted fault. Each of the following increases the probability of EDT aborted faults:

- Relatively aggressive compression (large chain-to-channel ratio)

- Large number of ATPG constraints
- Relatively small design

IDdq_restriction — A literal that displays all the undetected faults that the tool aborted while trying to satisfy the IDDQ restrictions.

INterrupt — A literal that displays all the undetected faults that the tool aborted because you interrupted the ATPG process with a Control-C.

MULtiple_clock_restriction — A literal that displays all the aborted, undetected faults due to multiple clock restrictions.

Ram_sequential — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between the ram_sequential patterns.

TRansition — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between the initial and final transition pattern.

WRite_pass_thru — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between write off and write on for RAM pass through patterns.

- >*file_pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >>*file_pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

report_aborted_faults

The following example displays the default summary of all the aborted faults for the design.

report_aborted_faults

```
10 backtrack
1  clock_control
2  mult_clock_restriction
```

This is the same as the command report_aborted_faults summary.

Example 2

This example displays all the aborted faults that are currently in the undetected fault list.

report_aborted_faults all

```
1    UO.AAB    /I$1/I$954/I4
0    UO.AAB    /I$1/I$952/OUT
1    UO.AAB    /I$1/I$952/OUT
0    EQ        /I$1/I$952/I3
0    EQ        /I$1/I$952/I2
0    EQ        /I$1/I$952/I1
0    EQ        /I$1/I$952/I0
1    UO.AAB    /I$1/I$952/I3
1    UO.AAB    /I$1/I$952/I2
1    UO.AAB    /I$1/I$952/I1
1    UO.AAB    /I$1/I$952/I0
```

Related Topics

[report_faults](#)
[set_abort_limit](#)
[set_atpg_limits](#)

report_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays all the current ATPG state restrictions and the pins on which they reside.

Usage

```
report_atpg_constraints [[{> | >>} file_pathname] [-Display {FLAt_schematic}]]
```

Description

Displays all the current ATPG state restrictions and the pins on which they reside.

The report_atpg_constraints command displays the pins and their state restrictions defined using the add_atpg_constraints command. The tool uses the state restrictions (constraints) during the ATPG process.

Arguments

- *>file_pathname*
An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.
- *>>file_pathname*
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.
- *-Display FLAt_schematic*
An optional switch and literal that specify to display the specific ATPG state of the primary input(s) as callouts in the Flat Schematic window.

Examples

The following example creates two ATPG pin constraints and then displays the information on those definitions:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
report_atpg_constraints

0 /I$135/Q (23)
1 and_b_in
```

Related Topics

[add_atpg_constraints](#)
[delete_atpg_constraints](#)

report_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays all the current ATPG function definitions.

Usage

```
report_atpg_functions [<> | >>] file_pathname]
```

Description

Displays all the current ATPG function definitions.

The report_atpg_functions command displays the definitions of the ATPG functions created using the add_atpg_functions command. You can use an ATPG function as an argument to the add_atpg_constraints command, which then lets you create state restrictions on pins that the tool uses during the ATPG process.

Arguments

- *>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- *>>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example creates two ATPG functions and then displays their definitions:

```
add_atpg_functions and_b_in and /i$143/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
report_atpg_functions

USER_AND and_b_in
  Input 0: /I$143/Q (27)
  Input 1: /I$141/Q (23)
  Input 2: /I$142/Q (25)
SELECT select_b_in
  Input 0: /I$144/Q (29)
  Input 1: /I$142/Q (25)
```

Related Topics

[add_atpg_constraints](#)

[add_atpg_functions](#)

[delete_atpg_functions](#)

report_atpg_timing

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Reports the current ATPG timing setup.

Usage

```
report_atpg_timing [<> | >>} file_pathname]
```

Description

Reports the current ATPG timing setup. For complete information, see “[Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual*.

The report uses the following format:

```
// ATPG timing = { ON | OFF }
// Timing source = { no timing |
//                   SDF minimum delay |
//                   SDF typical delay |
//                   SDF maximum delay }
// Hold PIs = { ON | OFF }
// Mask POs = { ON | OFF }
// Exclude inter-clock paths = { ON | OFF }
// Timing scale = { N/A | 1 ms | 1 us |
//                   1 ns | 1 ps }
// Slack margin for fault dropping = { OFF | slack_margin |
//                                         slack_margin_percentage% }
// Delay fault distribution function = { UNIFORM | HISTOGRAM
//                                         | EXPONENTIAL | INV_POLY }
// [ f(t)=function(t), (begin time = min and end time = max) ]
// Longest delay (rising, falling) = { max_rising, max_falling }
// Clock waveform = { UNDEFINED | USER_DEFINED }
//
// Clock Name           Test Clock           System Clock
//                     {Period, Offset, Width} {Period, Offset, Width}
// -----
// clock_name          num1,num2,num3        num1,num2,num3
...

```

- **ATPG timing** — The status of timing-aware ATPG.
- **Timing source** — The source of the timing information.
- **Hold PIs** — The state of holding PIs when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Mask POs** — The state of masking POs when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Exclude inter-clock paths** — The state of excluding inter-clock paths when doing static timing analysis for arrival time, propagation time, path delay, and slack.

- **Timing scale** — The time units used. The timing scale is set by the SDF file and defaults to 1 ns.
- **Slack margin for fault dropping** — The margin specified using the set_atpg_timing command.
- **Longest delay** — From static timing analysis.
- **Clock waveform** — The clock information specified by the set_atpg_timing command and reports the following:
 - **Clock Name**
 - **Test Clock** — The ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock). Timing-aware ATPG uses the test clock frequency.
 - **System Clock** — Normal operating frequency.

Arguments

- *>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- *>>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the timing-aware ATPG settings.

```
read_sdf top_worst.sdf
set_atpg_timing -clock test_clk_dd 16000 8000 8000
set_atpg_timing -clock test_clk_cc 8000 4000 4000
set_atpg_timing -clock test_clk_aa 4000 2000 2000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
report_atpg_timing
```

```
// ATPG timing = ON
// Timing source = SDF typical delay
// Time scale = 1 ns
// Slack margin for fault dropping = OFF
// Delay fault distribution function = EXPONENTIAL
// f(t) = 0.00158*e^(-0.0021t) + 4.94e-06, (begin time=0, end time=50000)
// Longest delay (rising, falling) = (50547.0000, 50547.0000)
// Clock waveform = USER DEFINED
//
// Clock Name Test Clock System Clock
// {Period, Offset, Width} {Period, Offset, Width}
// -----
// /test_clk_dd (4) 16000, 8000, 8000 16000, 8000, 8000
// /test_clk_cc (3) 8000, 4000, 4000 8000, 4000, 4000
// /test_clk_aa (2) 4000, 2000, 2000 4000, 2000, 2000
// /clk_in (6) 36000, 18000, 18000 36000, 18000, 18000
```

Related Topics

[read_sdf](#)

[set_atpg_timing](#)

report_attributes

Context: unspecified, all contexts

Mode: all modes

Creates a human readable report for the set of attributes based on the usage options.

Usage

Usage 1: Creates a report of attributes defined on a list of objects

```
report_attributes obj_spec [-predefined | -user_defined] [-silent] [-verbose]
```

Usage 2: Creates a report of attributes defined for specific object types

```
report_attributes -object_types type_list [-predefined | -user_defined] [-silent] [-verbose]
```

Usage 3: Creates a report for a specific attribute name

```
report_attributes -name attribute_name [-silent] [-verbose]
```

Usage 4: Creates a report of all registered attributes

```
report_attributes [-predefined | -user_defined] [-silent] [-verbose]
```

Description

Creates a human readable report for the set of attributes based on the usage options. In all usages, specifying the -predefined option restricts the report to only predefined attributes and specifying the -user_defined option restricts the report to only user-defined attributes.

Usage 1: Creates a tabular report for the attributes set on any object found in *obj_spec*. The attributes on an object that are inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object is reported in the list of attributes for the instances of that module. By default, attributes which have not been set on the objects are not listed in the report unless the attribute was registered with the -show_default option. If an object in *obj_spec* does not exist, an error message is displayed unless the -silent switch is specified.

Usage 2: Creates a tabular report for all attributes registered on the specified object types. These do not include the inherited attributes from one object type to another such as the module attributed inherited by the associated instances.

Usage 3: Creates a tabular detailed report for a specific attribute.

Arguments

- *obj_spec*

A required value used in Usage 1 that specifies a Tcl list of one or more object names or a collection of one or more objects.

- **-object_types type_list**

A required value used in Usage 2 to enumerate the object types for which the attribute report is requested. The supported object types are module, port, instance, pin and net.

- **-name attribute_name**

An optional switch and string pair that specify the name of the attribute whose information is to be reported. This argument is mutually exclusive with the *obj_spec* and -object_types *type_list* arguments.

- **-predefined | -user_defined**

An optional switch that constrains the report to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default no constraints are applied and both pre-defined and user-defined attributes are reported.

- **-silent**

An optional switch that specifies to suppress error messages if there are objects in the *obj_spec* that do not exist.

- **-verbose**

An optional switch that specifies to report the full list of possible attribute values. By default, only one line is dedicated to reporting the attribute values; additional values are truncated and replaced with “...”.

Examples

Example 1

The following example reports the attribute definitions using the report_attributes command:

```
report_attributes
Attribute Definition Report

Attributes defined for all objects:

Name      Object     Type      Class      Default    Constraints
-----
name      all        string    Pre-defined   -
speed     pin        integer   User-defined  200       100 to 500
added_by net        string    User-defined  John      John Patrick Susan
```

The same result would be achieved by executing:

```
report_attributes -object_types all
```

Example 2

The following example reports the attribute definitions for the pin object type:

```
report_attributes -object_types pin
```

Attribute Definition Report

Attributes defined for 'pin' objects:

Name	Object	Type	Class	Default	Constraints
<hr/>					
name	all	string	Pre-defined	-	-
speed	pin	integer	User-defined	200	100 to 500

Example 3**Scenario 1**

The following example reports the attribute definitions only for a list of pin objects:

report_attributes {u7/CLK u6/A2}

Attribute Definition Report

Attributes defined for object 'u7/CLK':

Name	Value	Inheritance
<hr/>		
base_name	u7	-
direction	input	-
is_valid	true	-
leaf_name	CLK	-
module_name	dff	-
name	u7/CLK	-
object_type	pin	-
parent_instance	-	-
parent_is_hard_module	true	-

Attributes defined for object 'u6/A2':

Name	Value	Inheritance
<hr/>		
base_name	u6	-
direction	input	-
is_valid	true	-
leaf_name	A2	-
module_name	xor3	-
name	u6/A2	-
object_type	pin	-
parent_instance	-	-
parent_is_hard_module	true	-

Scenario 2

This example reports the attributes for all scan elements of the class wrapper.

report_attributes [get_scan_elements -class wrapper]

Command Dictionary (R)
report_attributes

```
Attribute Definition Report
Attributes defined for object '/\cnt_s_reg[0] ':
Name          Value           Inheritance
-----
at_speed_control_group 2             -
cell_type      scan_cell        module
class          wrapper          -
clk_equation   -              -
clock_domain   inferred_occ_clock0
clock_edge     pos_edge         -
clock_group    -1              -
exists          true            -
has_children   false           -
has_no_definition false          module
id             3              -
is_created     false           module
is_hard_module true            module
is_modified    false           module
is_non_editable true           instance
is_non_editable_reason hard_module instance
is_unsaved     false           module
is_valid       true            -
language       tessonnt_lib    module
leaf_name      \cnt_s_reg[0]  instance
length         1              -
master_module_name sff            instance
module_name    sff            instance
name           /\cnt_s_reg[0]  -
object_type    scan_element    -
parent_instance -              instance
power_domain_island -            -
scan_in        SI              -
scan_out       Q               -
se_equation    se[0]           -
si_clock       inferred_occ_clock0
si_clock_edge  pos_edge        -
si_equation    -              -
simulation_function scan_cell    module
so_clock       inferred_occ_clock0
so_clock_edge  pos_edge        -
so_equation    -              -
state          usable           -
static_control_group -1           -
type           leaf_cell       -
unrolled_leaf_name \cnt_s_reg[0]  instance
unrolled_name  \cnt_s_reg[0]  instance
used          true            -
wrapper_type   output          -
wrapper_type_reason inferred_from_analysis -
```



```
Attributes defined for object '/\cnt_s_reg[1] ':
Name          Value           Inheritance
-----
at_speed_control_group 2             -
cell_type      scan_cell        module
class          wrapper          -
clk_equation   -              -
clock_domain   inferred_occ_clock0
```

clock_edge	pos_edge	-
clock_group	-1	-
exists	true	-
has_children	false	-
has_no_definition	false	module
id	4	-
is_created	false	module
is_hard_module	true	module
is_modified	false	module
is_non_editable	true	instance
is_non_editable_reason	hard_module	instance
is_unsaved	false	module
is_valid	true	-
language	tessent_lib	module
leaf_name	\cnt_s_reg[1]	instance
length	1	-
master_module_name	sff	instance
module_name	sff	instance
name	/\cnt_s_reg[1]	-
object_type	scan_element	-
parent_instance		instance
power_domain_island		-
scan_in	SI	-
scan_out	Q	-
se_equation	se[0]	-
si_clock	inferred_occ_clock0	-
si_clock_edge	pos_edge	-
si_equation		-
simulation_function	scan_cell	module
so_clock	inferred_occ_clock0	-
so_clock_edge	pos_edge	-
so_equation		-
state	usable	-
static_control_group	-1	-
type	leaf_cell	-
unrolled_leaf_name	\cnt_s_reg[1]	instance
unrolled_name	\cnt_s_reg[1]	instance
used	true	-
wrapper_type	output	-
wrapper_type_reason	inferred_from_analysis	-

Example 4

The following example reports the attribute definitions for the speed attribute:

report_attributes -name speed

Attribute Definition Report

Name: Speed
Description:

This attribute is used to hold the target frequency that the pin is allowed to have.
The allowed frequency range of 100 to 500 was chosen to match the technology spec.

Global Option Settings:

Setting	Value
Class	User-Defined
Type	int
Default	200
Show_Default	false
Minimum	100
Maximum	500
Object	pin

Specific 'pin' Type settings:

Name	Value
dense	false
export_to_netlist	off
save_in_flat_model	off
preserve_boundary_in_flat_model	off

Example 5

The following example reports the possible cell_type attribute values with the -verbose switch both enabled and disabled.

report_attribute -name cell_type

Attribute Definition Report

Name:

cell_type

Description:

This is a library model attribute

Setting	Value
Class	Pre-Defined
Value_Type	string
Enum	and buffer clock_and clock_buffer ...

GUI.Strings	cell_type = %
Object.Types	module

Specific 'module' Object Type settings:

Name	Value
export_during_write	auto
preserve_boundary_in_flat_model	off
applies_to_child_instances	on
display_in_gui	off
gui_marking_index	1

report_attribute -name cell_type -verbose

Attribute Definition Report

Name:

cell_type

Description:

This is a library model attribute

Setting	Value
Class	Pre-Defined
Value_Type	string
Enum	and buffer clock_and clock_buffer clock_gating_and clock_gating_or clock_inverter clock_mux clock_or dff inverter latch mux nand nor or pad scan_cell synchronizer_cell xor

GUI.Strings	cell_type = %
Object.Types	module

Specific 'module' Object Type settings:

Name	Value
export_during_write	auto
preserve_boundary_in_flat_model	off
applies_to_child_instances	on
display_in_gui	off
gui_marking_index	1

Related Topics

[get_attribute_list](#)
[get_attribute_option](#)
[get_attribute_value_list](#)
[register_attribute](#)
[set_attribute_options](#)
[unregister_attribute](#)

report_bisr_repair_register_icl_instances

Context: all contexts

Mode: all modes

Reports a list of BISR repair registers found in the design.

Usage

```
report_bisr_repair_register_icl_instances [-power_domain_group pdg_list]
```

Description

Reports the list of available memory BISR repair registers found in the design, providing the ICL instance name, design instance name and power domain group for each register. You can use the -power_domain_group argument to specify a listing of repair registers within specific power domain groups.

You must have already inserted the ICL modules with the [process_dft_specification](#) command and extracted the top-level ICL description with the [extract_icl](#) command before using report_bisr_repair_register_icl_instances.

Arguments

- **-power_domain_group pdg_list**

An optional argument and value pair that specifies one or more power domain groups from which the report lists available memory BISR repair register instances.

The *pdg_list* value supports [glob](#) expressions to aid in defining the desired list. For example, providing the argument -power_domain_group [list pdg*] will report the BISR register ICL instances for all registers that are on a power domain group that begins with “pdg”.

Examples

The following example reports the list of available memory BISR repair register instances on power domain groups “-” and “pdgA”:

```
report_bisr_repair_register_instances -power_domain_group [list - pdgA]

// BISR Repair Registers
// -----
// ----- Power Domain Group      ICL Instance          Design Instance -----
// ----- -
// -           core_blkA_i1_mem6_bisr_inst    core/blkA_i1/mem6_bisr_inst
// pdgA        core_blkA_i2_mem6_bisr_inst    core/blkA_i2/mem6_bisr_inst
// -           core_blkB_i1_memA_bisr_inst    core/blkB_i1/memA_bisr_inst
//
```

report_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the currently defined BIST capture ranges.

Usage

```
report_bist_capture_ranges
```

Description

The report_bist_capture_ranges command displays a list of the capture ranges that are currently defined as a result of the [add_bist_capture_range](#) command. A BIST capture range is defined by a named capture procedure that is stored in the test procedure file, and a range of patterns that will use this capture procedure. The complete range of patterns reported by the report_bist_capture_ranges command typically represents a small subset of the entire pseudo-random pattern space, and is applied to the complete pattern space by repeating the sequence until all the pseudo-random patterns have been simulated.

Arguments

None

Examples

The following example shows the output from the report_bist_capture_ranges command.

```
report_bist_capture_ranges

// List of currently defined Logic BIST capture ranges
// Procedure name    from pattern   to pattern
// -----  -----
// cap1              0            14
// cap2              15           15
//
// NOTE: This list defines the active named capture range during pseudo-
// random pattern simulation. The tool automatically restarts at the top
// of the list when the pattern counter matches the final "to pattern"
// value.
```

Related Topics

[add_bist_capture_range](#)

[delete_bist_capture_ranges](#)

report_black_boxes

Context: all contexts

Mode: setup, analysis

Displays information on blackboxes and undefined models.

Usage

```
report_black_boxes
  {{-INStances [ins_pathnames]} | {-Modules [module_names]} | -All | -Undefined}
  [-INClude_pins]
  [>|>>} file.pathname]
```

Description

Displays information on blackboxed modules, instances, and undefined modules.

The report_black_boxes command reports on the status of any instance-based and/or module-based blackboxes, or undefined models that are not yet blackboxed. The tool displays the string MODULE/MODEL/INSTANCE followed by the name in the following format:

```
// Black Boxed Modules and Instances:
// Default tie value = X
// MODULE:
// MODEL:
// INSTANCE:
```

Displaying Pins

When the optional -INClude_pins switch is specified, the tool displays a list of pins as in the following example:

```
// Black Boxed Modules and Instances:
// Default tie value = X
// MODULE: 'foo'
//   USER: Output pin 'b' tied to X
// MODEL: xor02
//   USER: Output pin 'Y' tied to X
// INSTANCE: 'hal'
//   USER: Output pin 's' tied to X
//   USER: Output pin 'c_out' tied to X
```

Arguments

- **-Instances [ins_pathnames]**

A switch and optional string that specify for the tool to display information on instance-based blackboxes. The *ins_pathnames* argument can be a Tcl list of one or more instances, or a collection of one or more instances. If you do not specify an *ins_pathname*, the tool displays information on all instance-based blackboxes. If you specify an *instance pathnames*, it reports on that single, instance-based blackbox.

- **-Modules** [*module_names*]

A switch and optional string that specify for the tool to display information on module-based blackboxes. The *module_names* argument can be a Verilog module or Tessent Cell library model, a Tcl list of modules, or a collection of modules. If you do not supply a *module_name*, the tool displays information on all module-based blackboxes. If you specify a *module_name*, it reports on that single, module-based blackbox.

- **-All**

A switch that specifies for the tool to display information on all defined blackboxes, as well as undefined models. This is the command default.

- **-Undefined**

A switch that specifies for the tool to display information on undefined models that have not yet been blackboxed. Use this switch to determine whether your design is complete, or is missing library models. If you intend to blackbox undefined models, this report allows you to verify that only the intended models are undefined.

- **-INClude_pins**

An optional switch that specifies reporting pins for defined blackboxes. See “[Displaying Pins](#).”

- **>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- **>>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example defines module- and instance- based blackboxes then reports on them.

```
add_black_box -module halfadder -pin s 1
add_black_box -instance final_carry -pin Y 0
report_black_box -include_pins

// Black Boxed Modules and Instances:
// Default tie value = X
// MODULE: 'halfadder'
//     USER: Output pin 's' tied to 1
//     USER: Output pin 'c_out' tied to x
// INSTANCE: 'final_carry'
//     USER: Output pin 'Y' tied to 0
```

Related Topics

[add_black_boxes](#)

[delete_black_boxes](#)

report_boundary_scan_port_options

Context: dft, patterns

Mode: all modes

Displays a table listing all of the ports with their inferred or specified cell_options.

Usage

```
report_boundary_scan_port_options
```

Description

Displays a table listing all of the ports with their inferred or specified cell_options as specified by the [set_boundary_scan_port_options](#) command.

When the design level is physical_block or sub_block, as specified using the [set_design_level](#) command, all ports not listed in the “[set_boundary_scan_port_options -pad_io_ports](#)” list are inferred as dont_touch. For more information on the dont_touch, see [Table 10-4](#) for a description of these values.

Arguments

None.

Examples

The following example runs the report_boundary_scan_port_options command on a design in which the level was set to physical_block using the set_design_level command.

```
set_design_level sub_block
set_boundary_scan_port_options -pad_io_ports [get_port gpio*]
set_boundary_scan_port_options gpio00 -cell_options clock
report_boundary_scan_port_options

// Pad io pins : gpio00 gpio01 gpio02 gpio03 gpio04 gpio05 gpio06 gpio07
//
// Port name          Options
// -----
// gpio00             clock
// gpio01             normal
// gpio02             normal
// gpio03             normal
// gpio04             normal
// gpio05             normal
// gpio06             normal
// gpio07             normal
// clk                dont_touch
// ina                dont_touch
// outa               dont_touch
```

Related Topics

[set_boundary_scan_port_options](#)

[get_boundary_scan_port_option](#)

report_bus_data

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the bus data information for either an individual bus gate or for the buses of a specific type.

Usage

```
report_bus_data type [{>|>>} file.pathname]
```

Description

Displays the bus data information for either an individual bus gate or for the buses of a specific type.

The report_bus_data command displays the following bus information:

- Instance name
- Gate identification number
- Contention handling (pass, bidi, fail, or abort)
- Type of bus (strong or weak)
- Number of drivers on the bus
- Any learned behavior of the bus.

The design rule that checks for bus contention mutual exclusivity is rule E10. For more information on rule E10, refer to the “[Extra Rules \(E Rules\)](#)” section.

If you enable the learn reporting by using the [set_learn_report](#) command, the tool provides the following two additional lines of information with the report_bus_data command:

- Information on whether or not the bus is capable of being set to an X, set to a Z, or having multiple drivers turned on.
- A list of drivers and their corresponding gate type. Drivers that are equivalent have a gate type of EQ.

The design rule that checks to see if there is any possible input combination that can force a bus into the high-impedance state (Z) is rule E11. For more information on rule E11, refer to the “[Extra Rules \(E Rules\)](#)” section.

Arguments

- **type**

A required literal or integer that specifies the type of bus for which you want the tool to display information. The choices for the **type** argument are as follows:

gate_id# — An integer that specifies the gate identification number whose bus data you want to display.

All — A literal that displays the bus data for all buses.

Weak — A literal that displays the bus data for the weak buses.

Strong — A literal that displays the bus data for the strong buses.

Dominant — A literal that displays the bus data for the final bus of every set of cascaded buses.

NONDominant — A literal that displays the bus data for all but the final bus in every set of cascaded buses.

Pass — A literal that displays the bus data for the buses that passed the contention mutual exclusivity checking.

Bidi — A literal that displays the bus data for the bidirectional buses that have possible contention problems. For the tool to place a bus in this category, the bidirectional pin must have only a single tri-state driver.

Fail — A literal that displays the bus data for the buses that failed the contention mutual exclusivity checking.

Abort — A literal that displays the bus data for the buses that aborted contention mutual exclusivity checking.

Buf — A literal that displays the bus data for all buses that have the buffer learned behavior.

Xor — A literal that displays the bus data for all buses that have the exclusive OR learned behavior.

Mux — A literal that displays the bus data for all buses that have the multiplexer learned behavior.

AND — A literal that displays the bus data for all buses that have the AND learned behavior.

OR — A literal that displays the bus data for all buses that have the OR learned behavior.

POSS_Mult_dr_on — A literal that displays the bus data for all buses that have a possibility of having multiple drivers turned on at the same time.

POSS_X — A literal that displays the bus data for all buses that have a possibility of being at an unknown state.

POSS_Z — A literal that displays the bus data for all buses that have a possibility of being at the high-impedance state.

Histogram — A literal that displays a summary of information identifying the number of buses that are in each of the possible categories.

ZPass — A literal that displays the bus data for the buses that pass the E11 design rule.

ZFail — A literal that displays the bus data for the buses that fail the E11 design rule check.

ZAbort — A literal that displays the bus data for the buses that abort the E11 design rule check.

- *>file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file.pathname*.

- *>>file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

Example 1

The following example displays the information on a specific bus gate—an inverter (INV):

```
report_bus_data 31
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
    Bus Drivers: 30(SW) 28(SW)
```

Example 2

The following example first enables access to the static learning data, then displays both the learned information and the bus data on a specific bus gate, again, an inverter (INV):

```
set_learn_report on
report_bus_data 31
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
    Learn Data : poss_X=no, poss_Z=no, poss_mult_drivers_on=no
    Bus Drivers: 30(SW) 28(SW)
```

Related Topics

[set_learn_report](#)

report_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Displays the defined EDT bypass scan chain connections.

Usage

```
report_bypass_chains [-All_blocks | {-Edt_chain chain_name} |  
{-Bypass_chain chain_number}] [{>|>>} file_pathname]
```

Description

Displays the defined EDT bypass scan chain connections. If issued from Setup mode, the report_bypass_chains command lists scan chain connections for bypass scan chains defined using the [set_bypass_chains](#) command.

If issued from analysis mode, all bypass scan chain connections are reported, whether defined by the tool or with the set_bypass_chains command.

You can display the chain connection information for all bypass chains, or optionally include the -Edt_chain or -Bypass_chain switch to display only the information for a particular bypass chain. The order, left to right, in which the display lists the scan chains in a particular bypass chain is the order in which the chains are series-connected from channel input to channel output.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT bypass chains in every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

 This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if

it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-Edt_chain *chain_name***

An optional switch and string pair that specify to display the scan chain connections for only the EDT bypass chain that includes the EDT scan chain, *chain_name*. You can use this switch to see which bypass chain or scan channel is associated with a particular scan chain.

- **-Bypass_chain *chain_number***

An optional switch and positive integer pair that specify to display the scan chain connections for the EDT bypass chain, *chain_number*. Use this switch to list the scan chain connections for a particular EDT bypass chain. The *chain_number* is the same as the number of the scan channel connected to the bypass chain.

- **> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- **>> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

Assume a design has eight scan chains named “chain1” through “chain8” and that the chains have been incorporated into two EDT bypass channels configured from two EDT scan channels. The following example displays the scan chain connection information for the EDT bypass chain containing chain3. You can see from the report that chain3 is part of a bypass chain connected to scan channel 2.

```
report_bypass_chains -edt_chain chain3
//      Reporting user-defined bypass chains:
//
//      Bypass#      EDT chains
//      -----      -----
//          2          chain2, chain3, chain4, chain7, chain8
```

Example 2

The following example displays the connection information for EDT bypass chain 1 (scan channel 1).

```
report_bypass_chains -bypass_chain 1
//      Reporting user-defined bypass chains:
//
//      Bypass#      EDT chains
//      -----      -----
//          1          chain1, chain5, chain6
```

Again, notice that the EDT bypass chain number (Bypass# in the display) is the same as the number of the scan channel to which the bypass chain is connected.

Related Topics

[reset_bypass_chains](#)

[set_bypass_chains](#)

report_capture_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays any special data capture handling currently in use.

Usage

```
report_capture_handling [List | Summary | SSources | SInks | Gates]  
[{> | >>} file_pathname]
```

Description

Displays any special data capture handling currently in use. To change how the tool handles data capturing on level-sensitive and trailing-edge state elements, use the add_capture_handling command. If you have not set up any special data capture handling, then the report_capture_handling command does not generate a report.

Arguments

- List
A literal that displays the handling settings (old, new, or X), whether the element is a sink or source, the instance pathname, and the gate identification number. This is the default.
- Summary
A literal that displays the handling settings for both level-sensitive and trailing-edge state elements, the number of sources, the number of primitive gates in the flattened netlist between source and sink points, and the number of sinks.
- Sources
A literal that displays the gates that have source-point, special data capture handling.
- SInks
A literal that displays the gates that have sink-point, special data capture handling.
- Gates
A literal that displays the identification numbers of all the primitive gates between the source and sink points.
- >file_pathname
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- >>file_pathname
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example sets up the data capture handling for one gate, and then issues the set_capture_handling command to identify the associated sinks for that source:

```
add_capture_handling new 1158 -source
set_capture_handling
```

The following set of commands show the different formats for the available reports:

```
report_capture_handling
```

```
NEW Source /I_1_16/DF0/ (1158)
```

```
report_capture_handling list
```

```
NEW Source /I_1_16/DF0/ (1158)
```

```
report_capture_handling summary
```

```
Capture handling summary: LS=OLD, TE=NEW, #sources=1,
#int_gates=14, #sinks=2
```

```
report_capture_handling source
```

```
Source list : 1158-X
```

```
report_capture_handling sink
```

```
Sink list : 1160-X 1165-X
```

```
report_capture_handling gates
```

```
Int_gate list: 13 14 39 40 51 52 53 61 62 63 68 69 79 86
```

Related Topics

[add_capture_handling](#)

[set_capture_handling](#)

report_capture_procedures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the cyclized information for a specified named capture procedure.

Usage

```
report_capture_procedures [-Clock_sequence] [-Created_capture_procedure |  
-Loaded_capture_procedure | -Enabled | -Disabled | capture_procedure_name...] [  
[-All | -Summary | -Internal | -External] [-Display Wave] [{> | >>} file.pathname]
```

Description

Displays the cyclized information for a specified named capture procedure. Cyclizing is a processing step performed prior to ATPG that merges internal mode cycles when needed to optimize them and/or expand other internal mode cycles to ensure correct simulation results.

For example, when a named capture procedure contains a cycle that pulses multiple clocks at different times, the cycle must be expanded into multiple cycles (one for each clock pulse) in order to simulate it. The cyclized information shows the cycles ATPG will actually use when creating patterns, after any merging or expansion of cycles. The display also includes comments about the timing of the cycles and where the at-speed sequences begin and end to aid your understanding.

The displayed information resembles the original procedure as defined in the test procedure file, but it does not have any timeplates. Therefore, it will not work if copied and pasted as is into a dofile; you must add timeplates. To display the procedure as it actually exists in the test procedure file, use the [report_procedures](#) command.

The number of cycles reported is the sequential depth used during test generation. The display of merged or expanded cycles is illustrated in the examples. See also “[Named Capture Procedures Display](#)” in the *Tessent Scan and ATPG User’s Manual*.

Note

 If the named capture procedure is modified to cyclize it for ATPG, the modifications are internal only; the named capture procedure is not altered in the test procedure file.

Arguments

- **-Clock_sequence**
An optional switch that displays the clock sequence of the named capture procedures. The scan load cycle(s), at-speed cycle(s), and the number of patterns the named capture procedure is associated with are also displayed.
- **-Created_capture_procedure**
An optional switch that displays only the created named capture procedures.

- **-Loaded_capture_procedure**
An optional switch that displays only the named capture procedures loaded from the read_procfile command or from the procedure file associated with the scan group definition.
- **-Enabled**
An optional switch that displays only enabled named capture procedures.
- **-Disabled**
An optional switch that displays only disabled named capture procedures.
- ***capture_procedure_name***
An optional, repeatable string that displays only the specified named capture procedures. By default, information for all named capture procedures is displayed.
- **-All**
An optional switch that displays information for both the internal and external modes of the named capture procedure. This is the default.
- **-Summary**
An optional switch that displays only summary information about the named capture procedure.
- **-Internal**
An optional switch that displays information for only the internal mode of the named capture procedure.
- **-External**
An optional switch that displays information for only the external mode of the named capture procedure.
- **-Display Wave**
An optional switch that displays the specified capture procedure in the Wave window of DFTVisualizer. This is for optimized NCP's. Use the [report_procedures](#) command to display unoptimized NCP's.
- **> *file.pathname***
An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file.pathname*.
- **>> *file.pathname***
An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file.pathname*.

Examples

Example 1

The following example displays summary information for the named capture procedures *nam_cap1*, *nam_cap2*, *my_cap1*, and *my_cap2*, including the scan load cycle(s), the at-speed

cycle(s), and the number of patterns with which the named capture procedures are associated. It also indicates *nam_cap1* and *nam_cap2* were loaded from an external file and *my_cap1* and *my_cap2* were internally created.

report_capture_procedures -summary

capture_proc_name	cycles	at_speed_cycle	active	source	# associated patterns	
					internal	external
nam_cap1	1	1	no	loaded	0	-
nam_cap2	1	1	yes	loaded	10	-
my_cap1	2	{1 to 2}	yes	created	5	-
my_cap2	2	{2 to 2}	no	created	0	-

Example 2

The following example displays the clock sequence of the named capture procedures *capture1*, *capture2*, *user_capture_proc*, and *user_capture_proc_1* in addition to the summary information.

report_capture_procedures -clock_sequence

capture_proc_name	cycles	at_speed_cycle	active	source	# associated patterns		capture_clock_sequence
					internal	external	
capture1	1	1	yes	loaded	6	-	S(c4,c3,c2,c1,c0)
capture2	1	1	yes	loaded	8	-	S(c9,c8,c7,c6,c5)
user_capture_proc	3	{1 to 3}	yes	created	0	-	S(c0)(c1,c0)(c3,c2)
user_capture_proc_1	3	{1 to 3}	yes	created	0	-	S(c0),(c1,c0)(c3,c2)

Example 3

The following example displays information for all the named capture procedures (*capture1*, *capture2*, and *my_cap1*). The first line displayed for each named capture procedure indicates if the procedure is currently enabled for test generation (ACTIVE) or disabled (INACTIVE). You can enable or disable capture procedures with the [set_capture_procedures](#) command.

report_capture_procedures

```
procedure capture "capture1" = // (INACTIVE) (loaded capture procedure)
    cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure PO)
        condition "/inst_0/clkenhold/reg_sco/" 1 ;
        force "/clock[4]" 0 ;
        force "/clock[3]" 0 ;
        force "/clock[2]" 0 ;
        force "/clock[1]" 0 ;
        force "/clock[0]" 0 ;
        force "/clear" 0 ;
        force "/scan_en" 0 ;
        force "/test_clk" 0 ;
        force_pi ;
        measure_po ;
        pulse "/clock[1]" ;
        pulse "/clock[0]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "capture1"
```

Command Dictionary (R)
report_capture_procedures

```
procedure capture "capture2" = // (ACTIVE) (loaded capture procedure)
    cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure PO)
        condition "/inst_0/clkenhold/reg_sco/" 0 ;
        force "/clock[4]" 0 ;
        force "/clock[3]" 0 ;
        force "/clock[2]" 0 ;
        force "/clock[1]" 0 ;
        force "/clock[0]" 0 ;
        force "/clear" 0 ;
        force "/scan_en" 0 ;
        force "/test_clk" 0 ;
        force_pi ;
        measure_po ;
        pulse "/clock[4]" ;
        pulse "/clock[3]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "capture2"

procedure capture "my_cap1" = // (ACTIVE) (created capture procedure)
    cycle = slow // cycle 1 starts at time 0 (Force PI)
        force "/clock[4]" 0 ;
        force "/clock[3]" 0 ;
        force "/clock[2]" 0 ;
        force "/clock[1]" 0 ;
        force "/clock[0]" 0 ;
        force "/clear" 0 ;
        force "/scan_en" 0 ;
        force "/test_clk" 0 ;
        force_pi ;
        measure_po ;
        pulse "/clock[4]" ;
        pulse "/clock[3]" ;
    end ; // cycle 1
    cycle = // cycle 2 starts at time 50 (Begin_AC_test) (Force PI) (Measure PO)
        condition "/inst_0/clkenhold/reg_sco/" 0 ;
        force "/clock[4]" 0 ;
        force "/clock[3]" 0 ;
        force "/clock[2]" 0 ;
        force "/clock[1]" 0 ;
        force "/clock[0]" 0 ;
        force "/clear" 0 ;
        force "/scan_en" 0 ;
        force "/test_clk" 0 ;
        force_pi ;
        measure_po ;
        pulse "/clock[2]" ;
        pulse "/clock[1]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "my_cap1"
```

Example 4

The following example displays how the tool will use the cycle during ATPG (the command's output is excerpted for brevity):

report_capture_procedures -internal

```
...
Mode internal =
    Cycle = // cycle 1 starts at time 0 (Begin_AC_test)
              // (Force PI)
        Force_pi ;
        Pulse /clk1 ;
    End ; // cycle 1 at time 20
    Cycle = // cycle 2 starts at time 20
        Pulse /clk2 ;
    End ; // cycle 2 at time 50
...

```

The display shows that for this particular design and cycle definition, the tool expands the one cycle from the test procedure file into two cycles internally. Notice that the combined time of the two cycles is the same as the period of the original cycle in the test procedure file. The tool will automatically adjust the sequential depth used during ATPG to accord with the resultant change in number of cycles.

Example 5

The next example shows a case where the tool merges two cycles described in the test procedure file into one cycle for ATPG. The relevant excerpts from the test procedure file are shown first; the example then displays the cyclized data for the three cycles.

```
mode_internal =
...
timeplate tp2 =
    force_pi 0;
    pulse clk 10 10;
    period 50;
end;
cycle =
    force_tms 0;
    pulse clk;
end;
cycle =
    force_tms 0;
    pulse clk;
end;
...

```

report_capture_procedures -internal

```
...
Mode internal =
    Cycle = // cycle 1 starts at time 0 (Begin_AC_test)
        // (Force PI)
        Force_pi ;
        Force /tms 0 ;
        Pulse /clk ;
    End ;      // cycle 1 at time 100
    Cycle = // cycle 2 starts at time 100
        Force /tms 0 ;
        Pulse /clk ;
    End ;      // cycle 2 at time 150
```

Related Topics

[create_capture_procedures](#)
[report_procedures](#)
[report_timeplate](#)
[set_capture_procedures](#)

report_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the constrained cells.

Usage

```
report_cell_constraints [-Scan_cell | -Non_scan_cell]
[{-Chain chain_name} | {-Clock clock_name}]
[{> | >>} file.pathname]
```

Description

The report_cell_constraints command displays a list of all the scan cells (also non-scan cells for DX and SX constraints) that you previously constrained to a constant value using the [add_cell_constraints](#) command. The display may consist of up to seven columns, as follows (columns that contain no data for all cell constraints are not displayed):

- Constraint value — Specifies the constraint value.
- Chain name — Specifies the scan chain name if the constraint is on a scan cell. If the constraint is on a non-scan cell, then this column is blank.
- Cell position — Specifies the position in the scan chain if the constraint is on a scan cell. If the constraint is on a non-scan cell, then this column is blank.
- Cell instance name — Specifies the cell instance of the constraint. This column is blank unless you originally specified the constraint with a *pin.pathname*.
- Gate instance name — Specifies the gate instance of the constraint. This column is blank unless you originally placed the constraint on a non-scan cell using the -Clock switch.
- Constraint properties — Displays constraint properties, as applicable, from the following list:
 - Internal — Constraint was applied by the tool automatically in response to certain DRC results
 - Whole Chain — Constraint was applied using the -Chain switch
 - Drc C6 — Constraint was applied using the “-Drc C6” argument combination.
 - Clock — Constraint was applied using the -Chain switch
 - Drc D8 Warn — Constraint was applied by the tool automatically to a D8 failing master cell when “set_drc_handling d8 warning” was in effect for DRC. See “[D8](#)” for more information.
 - Dynamic — Constraint was applied in analysis mode
 - Static — Constraint was applied in setup mode

- Pattern scope — Specifies the pattern scope of the constraint: whether the tool uses the constraint only when creating scan patterns, or uses the constraint for both scan and chain pattern creation.

Arguments

- **-Scan_cell**
An optional switch that specifies to report_cell_constraints at scan cells only. This switch is valid in analysis modes only.
- **-Non_scan_cell**
An optional switch that specifies to report_cell_constraints at non-scan cells only. This switch is valid in analysis modes only.
- **-CHain *chain_name***
An optional switch and string pair that specifies to report_cell_constraints added to the scan chain, chain name, by an earlier add_cell_constraints -Chain command. This switch is valid in analysis modes only.
- **-Clock *clock_name***
An optional switch and string pair that specifies to report the cell constraints added by an earlier add_cell_constraints -Clock command to latches and flip-flops controlled by the clock, *clock_name*. This switch is valid in analysis modes only.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example begins in Setup mode and constrains a single scan cell to be at a constant zero for the generation of all patterns. The example then displays current cell constraints, leaves Setup mode, triggering the tool's automatic DRC process and again reports the cell constraints:

```
add_cell_constraints sdff1/q c0 -all_patterns
report_cell_constraints

// constraint cell_instance    constraint      pattern scope
//   value       name        properties
// -----
//   C0          sdff1/q     (Static)    Chain + Scan Patterns

set_system_mode analysis
report_cell_constraints
```

```
// const chain cell      cell_inst constraint      pattern scope
// value   name  pos       name    properties
// ----- -----
// C0      c1     2        sdff1/q  (Static)  Chain + Scan Patterns
```

Notice that once the tool has identified the scan chains during the DRC process, the report identifies the chain and cell position of the constrained scan cell.

Example 2

The next example adds DX constraints on all flip-flops clocked by clk2. You can see from the cell constraint report that these flip-flops are not scan cells (no chain or cell position). Notice also that the gate instances are reported for them since each flip-flop is a standalone gate, not part of a scan cell.

```
add_cell_constraints -clock /clk2 -type flop dx
report_cell_constraints

// const chain cell      cell_inst gate_inst constraint      pattern scope
// value   name  pos       name    name    properties
// ----- -----
//      DX                                /dff6   (Clock)  Scan Patterns only
//      DX                                /dff5   (Clock)  Scan Patterns only
// ----- -----
// C0      c1     2        sdff1/q  (Static)  Chain + Scan Pattern
```

Related Topics

[add_cell_constraints](#)

[delete_cell_constraints](#)

report_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays a list of either all cell models or the DFT library models associated with the specified cell type.

Usage

report_cell_models [-All | {-Type *cell_model_type*}]

Description

The report_cell_models command displays the cell models that you either added with the [add_cell_models](#) command, or described in the DFT library with the [cell_type](#) attribute.

Arguments

- -All
An optional switch that specifies to display all cell model definitions that you added with the [add_cell_models](#) command. This is the default.
- -Type *cell_model_type*
An optional switch and literal pair that specifies to display a listing of all the cell models of a particular type. The valid *cell_model_types* are as follows (with the minimum typing characters shown in uppercase):
 - INV** — A literal that specifies a one-input inverter gate.
 - And — A literal that specifies a two-input AND gate.
 - Buf — A literal that specifies a one-input buffer gate.
 - OR — A literal that specifies a two-input OR gate.
 - NAnd — A literal that specifies a two-input NAND gate.
 - NOr — A literal that specifies a two-input NOR gate.
 - Xor — A literal that specifies a two-input exclusive OR gate.
 - Mux — A literal that specifies a 2-1 multiplexer.
 - Scancell — A literal that specifies a cell with four input pins (clock, data, scan in, and scan enable), clocked scan cell with four inputs (clock, data, scan clock, and scan enable), or LSSD scan cell with five inputs (clock, data, scan in, master clock, and slave clock).
 - DFf — A literal that specifies a D flip-flop with two input pins (clock and data).
 - LATCH — A literal that specifies a D latch with two input pins (enable and data).

Examples

The following example displays a list of all added cell models:

```
add_clocks 0 clk
set_test_logic -set on -re on -clock on
set_system_mode analysis
report_scan_elements
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Topics

[add_cell_models](#)

[delete_cell_models](#)

[set_test_logic](#)

report_chain_masks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Reports the list of masked scan chains and their associated unload value.

Usage

```
report_chain_masks
```

Description

Reports the list of masked scan chains and their associated unload value.

Arguments

None

Examples

The following example adds several chain masks and then reports the added chain masks:

```
add_chain_masks chain2 chain4 chain5
report_chain_masks

// Masked scan chain Unload value Cell constraints
// -----
//      chain2          X          OX
//      chain4          X          OX
//      chain5          X          OX
```

The following example adds several chain masks, specifies load values, and then reports the added chain masks:

```
add_chain_masks chain2 -load_value 0
add_chain_masks chain4 -load_value 0
add_chain_masks chain5 -load_value 1
report_chain_masks

// Masked scan chain Load value Unload value Cell constraints
// -----
//      chain2          0          X          OX
//      chain4          0          X          OX
//      chain5          1          X          OX
```

Related Topics

[add_chain_masks](#)

[delete_chain_masks](#)

report_clock_controller_pins

Context: dft -logic_bist

Mode: setup, analysis

Reports the clock controller pins specified with the set_clock_controller_pins command.

Usage

```
report_clock_controller_pins [> | >>file.pathname]
```

Description

Reports the clock controller pins specified with the set_clock_controller_pins command.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- `>file.pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of `file.pathname`.
- `>>file.pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of `file.pathname`.

Examples

This example reports all clock_controller pins specified so far using the set_clock_controller_pins command.

```
report_clock_controller_pins
```

```
//
// PinType           Connection
// -----           -----
// Shift clock      clock_control_i1/SLOW_CLK clock_control_i2/
SLOW_CLK clock_control_i3/SLOW_CLK
// Scan enable     clock_control_i1/SCAN_EN clock_control_i2/
SCAN_EN clock_control_i3/SCAN_EN
// LBIST enable    clock_control_i1/LBIST clock_control_i2/LBIST
clock_control_i3/LBIST
// Capture procedure index NCP_INDEX_1/A NCP_INDEX_0/A
//
```

Related Topics

[set_clock_controller_pins](#)

[set_lbist_power_controller_options](#)

report_clock_controls

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays current configuration information for clock control definitions.

Usage

```
report_clock_controls
```

Description

Displays current configuration information for clock control definitions.

For more information, see “[Support for Internal Clock Control](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

None

Examples

The following example illustrates how clock control definitions are reported.

The following clock control is defined in a test procedure file.

```
clock_control clk =
    source_clock pll_clk;
    atpg_cycle 0=
        condition ctl_dff2/Q 1;
    end;
    atpg_cycle 1=
        condition ctl_dff1/Q 1;
    end;
end;
```

The Report Clock Control command reports the definition as follows:

```
report_clock_controls

CLOCK_CONTROL "/clk (6)" =
SOURCE_CLOCK "/pll_clk (4)";
    ATPG_CYCLE 0 =
        CONDITION "/ctl_dff2/ (56)" 1;
    END;
    ATPG_CYCLE 1 =
        CONDITION "/ctl_dff1/ (55)" 1;
    END;
END;
```

Related Topics

[set_clock_controls](#)

report_clock_domains

Context: dft -edt, patterns -scan

Mode: analysis

Displays all clocks and their corresponding clock domain numbers.

Usage

```
report_clock_domains [{-Compatible_clocks [-Details]} | {-Race_points clk1 clk2}]  
[{> | >>} file_pathname]
```

Description

During ATPG, if “set_clock_restriction domain_clock” is set, the tool may simultaneously turn on any subset of compatible clocks. This allows fault effects to be captured by more than one clock in the same cycle without risk of simulation mismatches due to clock skew. As a result, the test pattern count may be reduced and the fault coverage improved for the same clock sequential depth, compared with when only one clock is turned on in each cycle.

The report_clock_domains command lists defined clocks by groups based on their compatibility. Clocks are compatible (occupy the same clock domain) if capturing with them simultaneously can be performed reliably. Use this command to assess the overall clock compatibilities in your design, and to determine if using “set_clock_restriction domain_clock” would be beneficial.

Note

 The basic display enables you to quickly evaluate the overall clock compatibility of your design. It is summary information, however, and does not necessarily show the compatibility relationship between all possible clock pairings. That is, two clocks listed with different clock domains are not necessarily incompatible. Use the -Compatible_clocks switch if you need to check the compatibility information for a specific pair of clocks.

Arguments

- Compatible_clocks

An optional switch that includes in the display the pair-wise clock compatibility information the tool uses during ATPG. Because two clocks with different clock domain numbers are not necessarily incompatible, this switch enables you to determine precisely whether two clocks are compatible or not.

The tool treats two clocks as compatible if the clocks can be in their “on” state at the same time without causing clock skew problems or race conditions. To make this determination, the tool analyzes the clocking relationship between source and sink memory elements where the source is clocked by one clock and the sink is clocked by a different clock.

[Table 5-6](#) summarizes the conditions under which the tool determines the two clocks to be compatible (C) or incompatible (I).

Table 5-6. Clock Compatibility Summary

Source	Sink			
	LE ¹	TE ²	AH ³ (TE)	AL ⁴ (LE)
LE	I	C	C	I
TE	C	I	I	C
AH (LE)	I	C	C	I
AL (TE)	C	I	I	C

1. LE = Captures on PI clock's leading edge
2. TE = Captures on PI clock's trailing edge
3. AH = Active high, level sensitive element
4. AL = Active low, level sensitive element

- Details

An optional switch that displays the same compatibility information provided by the -Compatible_clocks switch, but instead of using an “x” in the display to mark each incompatible clock pair, use as the marker the number of state elements at which the tool determined the pair to be incompatible.

- Race_points *clk1 clk2*

An optional switch and pair of strings that lists the instances that cause the two clocks, *clk1* and *clk2*, to be incompatible. The tool includes the gate ID and type of state element (scan, non-scan or RAM) for each instance listed.

- >*file.pathname*

An optional redirection operator and pathname pair used at the end of the argument list or creating or replacing the contents of *file.pathname*.

- >>*file.pathname*

An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file.pathname*.

Examples

Example 1

The following example displays the defined clocks and their domains of the design. The numbers in parentheses are the unique gate identification numbers assigned to each gate during the model flattening process.

```
set_system_mode analysis
report_clock_domains
```

```
// No. | Clock name      Domain
// ---+-----+
//   1 | '/ck11m' (8)    1
//   2 | '/ck22m' (9)    1
//   3 | '/ck2_75m' (10)  2
//   4 | '/ck5_5m' (11)   2
//   5 | '/nreset' (39)   3
//   6 | '/scan_reset' (63) 4
//
// 6 clocks classified into 4 domains.
// This is only one of the possible clock classifications. To see all
// dependencies of clock pairs, use 'report_clock_domains
// -Compatible_clocks'
```

Example 2

The following example displays the clock domains in example 1 with added information about the possible clock pairs in the design. Notice that although the tool assigned the /scan_reset clock to a domain different from the others, it is actually incompatible only with clocks /ck2_75m, and /nreset. This is indicated by an “x” in the /scan_reset row, above the index number that corresponds to each of those two clocks.

report_clock_domains -compatible_clocks

```
// No. | Clock name      Domain | Clock compatibility
// ---+-----+-----+
//   1 | '/ck11m' (8)    1| .
//   2 | '/ck22m' (9)    1| ..
//   3 | '/ck2_75m' (10)  2| .x.
//   4 | '/ck5_5m' (11)   2| ....
//   5 | '/nreset' (39)   3| .xxx.
//   6 | '/scan_reset' (63) 4| ..x.x.
//
//          No. | 123456
//  . = Compatible (non-interacting) clock pair
//  x = Incompatible (interacting) clock pair
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

Example 3

The following example displays the same information as the preceding example, but directs the tool to substitute for each “x” the number of state elements at which the tool found the pair to be incompatible.

report_clock_domains -compatible_clocks -details

```
// No. | Clock name      Domain | Clock compatibility
// ---+-----+-----+
//   1 | '/ck11m' (8)    1| .
//   2 | '/ck22m' (9)    1| . .
//   3 | '/ck2_75m' (10)  2| . 3 .
//   4 | '/ck5_5m' (11)   2| . . . .
//   5 | '/nreset' 39)    3| . 6 1 6 .
//   6 | '/scan_reset' (63) 4| . . 8 . 4 .
// -----
//           No.| 1 2 3 4 5 6
// . = Compatible (non-interacting) clock pair
// x = Incompatible (interacting) clock pair
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

The report shows there are three state elements that cause clocks /ck2_75m and /ck22m to be incompatible. The next example lists the instance name, gate ID and type of element for each of these elements.

report_clock_domains -race_points /ck2_75m /ck22m

```
//     /ff30/ (57) non-scan
//     /ff31/ (58) non-scan
//     /ff31b/ (58) non-scan
// 3 state elements cause clocks "/ck2_75m" and "/ck22m" to be
// incompatible.
```

Example 4

This is an example of a clock domain report with a norace synchronous clock group:

report_clock_domains -compatible_clocks

```
//  No. | Clock Name  Domain | Clock Compatibility
//  ---+-----+-----+
//   1 | '/c1' (4)    1 | .
//   2 | '/c2' (5)    2 | x.
//   3 | '/c3' (6)    1 | .n.
//  -
//           No. | 123
// . = Compatible (non-interacting) clock pair
// x = Incompatible (interacting) clock pair
// n = Norace (synchronous group) specified between compatible clock pair (1)
// N = Norace (synchronous group) specified between incompatible clock pair (0)
//
// 3 clocks grouped into 2 domains.
// This is one of possible clock grouping that ATPG may use.
// Compatibility analysis is based on same-edge clock interaction.
```

Example 5

This is an example of a clock domain report with a race specified between an incompatible clock pair:

report_clock_domains -compatible_clocks

```
// No. | Clock Name Domain | Clock Compatibility
// -----+-----+-----+
//   1 | '/c1' (4)      1 | .
//   2 | '/c2' (5)      2 | r.
//   3 | '/c3' (6)      1 | ...
// -----+-----+-----+
//           No. | 123
// . = Compatible (non-interacting) clock pair
// x = Incompatible (interacting) clock pair
// R = Race specified between compatible clock pair (0)
// r = Race specified between incompatible clock pair (1)
//
// 3 clocks grouped into 2 domains.
// This is one of possible clock grouping that ATPG may use.
// Compatibility analysis is based on same-edge clock interaction.
```

Related Topics

[add_clocks](#)
[add_synchronous_clock_group](#)
[delete_clocks](#)
[set_clock_restriction](#)

report_clock_gating

Context: dft -scan, dft -test_points

Mode: analysis

Reports specified clock gating instances or modules in the design.

Usage

`report_clock_gating object_spec`

Description

Reports specified clock gating instances or modules in the design. Only clock gaters with unconnected or tied test enable pins are reported.

The report includes the following information for each clock gater:

- Unconnected Port — Specifies the unconnected test enable port that will be used to control the clock gater.
- Shift Control — Specifies the scan enable signal (or signals) that will be used to enable the clock gater during shift. The value is only available after issuing `analyze_scan_chains` or if there is an enable signal defined for the clock gater with the `set_clock_gating_enable` command.
- Capture Control — If the clock gater drives input or output wrapper cells, the column will display the appropriate mode enable signal name that will be ORed together with scan enable to guarantee correct wrapper chain operation. If the clock gater does not drive any input or output wrapper cells the column shows “Functional”, which means that the clock gater is not enabled during capture neither in internal nor external mode. The value is available only after issuing the `analyze_wrapper_cells` and `analyze_scan_chains` commands.
- Origin — Specifies how the tool first learned about the clock gater. The possible values are: User-specified, Cell library, Auto-identified.
- Status — Specifies the status of each clock gater. The possible values are:
 - Unconnected — Will get connected during scan insertion.
 - Hard module — The clock gater is inside a hard module and cannot be connected.
 - Excluded — Explicitly excluded by you with the “`set_clock_gating off`” command.
 - Non-scan — The clock gater does not drive any scannable cells.

Arguments

- *object_spec*

A Tcl list of one or more instances or modules or a collection of instances or modules.

Examples

The following example demonstrates the use of the report_clock_gating command:

report_clock_gating

```
report_clock_gating
-----
Clock Gating      Unconnected     Shift      Capture      Origin      Status
Instance          Port           Control    Control
-----
alwaysOn          -              scan_en   int_ltest_en User-specified Excluded
verilogCG         TE            scan_en   int_ltest_en Auto-identified Unconnected
drivesNonScan    TE            scan_en   ext_ltest_en Cell library Non-scan
cellCG           TE            scan_en   ext_ltest_en Cell library Unconnected
-----
```

Related Topics

[set_clock_gating](#)

report_clocks

Context: all contexts

Mode: setup, analysis

Displays a list of all user-defined clocks and SDC-defined clocks.

Usage

```
report_clocks [-All | -User | -Sdc] [-Display | [FLAt_schematic] | [HIEarchical_schematic] |  
[DAta] | [Wave] | ALl [APpend]] [{> | >>} file_pathname]
```

Description

Displays a list of all user-defined clocks and SDC-defined clocks. The report_clocks command lists all user-defined clocks created with the [add_clocks](#) command or all SDC-defined clocks created by an SDC file loaded with the [read_sdc](#) command or both.

Arguments

- **-All | -User | -Sdc**

A 1-of-3 choice of optional switches that report user-defined clocks or SDC clocks or both. User-defined clocks are created by the [add_clocks](#) command and SDC clocks are created by the SDC file loaded with the [read_sdc](#) command. The default option is -User. Note that the -Sdc switch is valid only when the flat model is available.

For information about the user-defined clocks report, refer to [Example 3](#). For information about the SDC clocks report, refer to [Example 4](#).

- **-Display | [FLAt_schematic] | [HIEarchical_schematic] | [DAta] | [Wave] | ALl} [APpend]**
An optional switch and repeatable literal that specify to display the instance(s) in the specified window; if Append is specified, the instances are added to the existing contents of the window instead of replacing the contents. The window choices are as follows:

FLAt_schematic — Specifies the Flat Schematic window. This is the default.

HIEarchical_schematic — Specifies the Hierarchical Schematic window.

DAta — Specifies the Data window.

Wave — Specifies the Wave window.

ALl — Specifies all supported windows.

- **> file_pathname**

An optional redirection operator and pathname pair for creating *or* replacing the contents of file_pathname.

- **>> file_pathname**

An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

Example 1

The following example adds two clocks to the clock list and then displays a list of the clocks:

```
SETUP> add_clocks 1 I1
SETUP> add_clocks 0 I2
SETUP> report_clocks

User-defined Clocks (2) :
=====
Sync and Async Source Clocks
=====

----- ----- ----- -----
Name Off State Constraints Internal
----- ----- ----- -----
'I1' 1 No
'I2' 0 No
```

Example 2

The following example adds two internal clocks to the clock list and merges them into a new primary input:

```
SETUP> add_clocks 1 u2/CLK u4/u5/u2/CLK -pseudo_port_name group_clk
// Note: Primary input 'group_clk' is added, merging 2 pins.

SETUP> report_clocks

User-defined Clock (1) :
=====
Sync and Async Source Clocks
=====

----- ----- ----- ----- ----- -----
Name Off State Constraints Internal Other Properties Pin(s)
----- ----- ----- ----- -----
group_clk 1 Yes Merged internal pin u2/CLK
u4/u5/u2/CLK
```

Example 3

The following example reports only user-defined clocks:

```
SETUP> add_clocks 1 u2/CLK u4/u5/u2/CLK -pseudo_port_name group_clk
// Note: Primary input 'group_clk' is added, merging 2 pins.
SETUP> add_clocks 0 I1 -period 50 ns
SETUP> add_clocks 1 I2 -pulse_always
SETUP> add_clocks I3 -pulse_in_capture -label clock_3
SETUP> report_clocks -user
```

User-defined Clocks (4) :							
Sync and Async Source Clocks							
Name	Label	Off State	Constraints	Internal	Period	Other Properties	Pin(s)
group_clk	group_clk	1			Yes	Merged internal pin u2/CLK u4/u5/u2/CLK	
I1	I1	-	Asynchronous	No	50.00ns		
I2	I2	1	Pulse always	No			
I3	clock_3	0	Pulse in capture	No			

The following explains the columns in the report:

- Name — Lists the name of the clock defined with the -pseudo_port_name switch or by default.
- Label — Lists the label name specified with the -label switch or assigned by default.
- Off State — Lists the off state of the clock as defined by the add_clocks command.
- Constraints — Lists the special constraint of the clock specified with the add_clocks command:
 - Pulse always — The clock is defined with the -pulse_always switch.
 - Asynchronous — The clock is defined with the -period switch.
 - Pulse in capture — The clock is defined with the -pulse_in_capture switch.
- Internal — Indicates whether the clock is defined on an internal pin.
- Period — Lists the clock period defined with the -period switch.
- Other Properties — Lists any additional properties of the clock.
- Pin(s) — Lists one or more internal pins that are connected to the clock.

Example 4

The following example reports only SDC clocks:

> report_clocks -sdc

SDC-defined Clocks (3) :					
Type	Add_clocks	Create_clock	Divide_by	Name	Nodes
Source	X	X		clk1x /clk1 (1)	
Generated			2	clk2x /CG/occicg_clk2x_top_inst/gclk (26)	
Generated			4	clk4x /CG/occicg_clk4x_top_inst/gclk (36)	

The report groups the generated clocks under the source clock from which they're derived. The following explains the columns in the report:

- Type— Lists the type of SDC clock, which is either a Source clock (from SDC `create_clock` command) or a Generated clock (from the SDC command `create_generated_clock`).
- Add_clocks — Indicates if the clock is also defined with the `add_clocks` command.
- Create_clock — Indicates when a source clock is defined using SDC `create_clock` command.
- Divide_by — Lists the `divide_by` ratio of the clock (from the SDC command “`create_generated_clock -divide_by`”).
- Name — Lists the name of the clock.
- Nodes — Lists the target nodes of the clock.

Example 6

The following example adds two internal clocks and then shows the output of the `report_clocks` command:

```
SETUP> add_clocks 0 clk1_inst/Y clk2_inst/Y clk3_inst/Y -pseudo_port_name clock_1x_int
SETUP> add_clocks 0 clk4_inst/Y clk5_inst/Y clk6_inst/Y -pseudo_port_name clock_2x_int
SETUP> report_clocks

User-defined Clocks (2):
-----
Name      Off State Constraints Internal Other Properties   Pin(s)
-----
'clk4_inst'  0       Yes           Merged internal pi  clk4_inst/Y
                                         clk5_inst/Y
                                         clk6_inst/Y
'clk5_inst'  0       Yes           Merged internal pin clk1_inst/Y
                                         clk2_inst/Y
                                         clk3_inst/Y
```

Example 7

The following example adds and then reports a clock branch:

```
SETUP> add_clocks u4/u5/u2/CLK -branch -label branch_2
SETUP> report_clocks

User-defined Clock (1):
Branch Clock
=====
Off State  Pin(s)      Label
-----
-          u4/u5/u2/CLK branch_2
```

Example 8

The following example uses the [set_clock_options](#) command to specify a TCK MUX insertion point using the -dft_inject_node switch, introspects the status, and reports the clocks:

```
SETUP> add_clocks 0 clk
SETUP> set_clock_options clk2 -dft_inject_node udff15/CLK
SETUP> set_clock_options clk3 -dft_inject_node udff20/CLK
SETUP> puts [get_name_list [get_clock_option clk2 -dft_inject_node]] udff15/CLK
SETUP> puts [get_name_list [get_clock_option clk3 -dft_inject_node]] udff20/CLK
SETUP> report_clocks

User-defined Clocks (3) :
=====
Sync and Async Source Clocks
=====
-----
Name      Off State   Constraints     Internal    Period      DFT Inject Node
-----
'clk'      0           Asynchronous   No          10.00ns    'clk'
'clk2'     -           Asynchronous   No          10.00ns    'dff15/CLK'
'clk3'     -           Asynchronous   No          10.00ns    'dff20/CLK'
```

Related Topics

[add_clocks](#)
[analyze_control_signals](#)
[delete_clocks](#)
[get_clocks](#)
[get_clock_option](#)
[read_sdc](#)
[set_clock_options](#)

report_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays defined scan chain connections for each EDT scan channel output.

Usage

```
report_compactor_connections [-All_blocks | {-CHANnel channel_number} |  
{-CHAIn chain_name} ] [{>|>>} file_pathname]
```

Description

Displays defined scan chain connections for each EDT scan channel output. If issued from Setup mode, the report_compactor_connections command lists scan chain connections for EDT scan channel outputs previously defined using the [set_compactor_connections](#) command.

If issued from analysis mode, all compactor connections are reported, whether defined by Tesson TestKompress or with the set_compactor_connections command.

You can display the chain connection information for all scan channel outputs, or optionally include the -Channel or -Chain switch to display the information only for a particular scan chain or channel.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tesson TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT scan channel outputs in every EDT block. You set the current EDT block using either the [add_edt_blocks](#) or [set_current_edt_block](#) command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tesson TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

 This switch is only for the top-level Pattern Generation Phase of modular Tesson TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tesson TestKompress flow, you do not need this switch.

An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays the information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-CHANnel *channel_number***
An optional switch and positive integer pair that specify to display the scan chain connections for only the scan channel identified by *channel_number*.
- **-CHAIn *chain_name***
An optional switch and string pair that specify to display the scan chain connections for only the scan channel whose compactor group includes *chain_name*. Use this switch to see which scan channel is associated with a particular chain.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

Assume a design has eight scan chains named “chain1” through “chain8”, whose outputs currently have user-specified connections through the compactor to two EDT scan channels. These user-defined connections are the result of previous `set_compactor_connections` commands.

The following example displays the compactor connection information for the chain3 scan chain. You can see from the report that chain3 is part of a compactor group that also includes chain4 and chain5; also that the outputs of these three chains are currently specified to be compacted into the output of scan channel number 2.

```
report_compactor_connections -chain chain3
```

```
//Reporting user-defined compactor connections:  
//  
//      Channel      Chains  
//      -----      -----  
//          2      chain3 - chain5  
//
```

Example 2

The following example looks up the connection information for scan channel number 1.

```
report_compactor_connections -channel 1
```

```
//Reporting user-defined compactor connections:  
//  
//      Channel      Chains  
//      -----      -----  
//          1      chain1, chain2, chain6, chain7, chain8  
//
```

Related Topics

[reset_compactor_connections](#)

[set_compactor_connections](#)

report_config_data

Context: unspecified, all contexts

Mode: all modes

Reports the content of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to a file using the write_config_data command.

Usage

```
report_config_data [config_object] [-in_wrapper wrapper_object_spec]  
    [-partition partition] [-show_unspecified]  
    [-levels integer / all] [-indent integer]  
    [-left_indent left_indent]
```

Description

Reports the contents of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to a file using the [write_config_data](#) command.

Like all report commands, you can use the greater than symbol “>” to redirect the output into a file.

Arguments

- ***config_object***

A required value that specifies the name of a configuration element or a collection containing one configuration element. When config_object is a collection, the -in_wrapper and -partition options cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the config_object value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)” and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- **-in_wrapper *wrapper_object_spec***

An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The names in config_object are relative to the specified parent wrapper.

- **-partition *partition***

An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications ([DftSpecification](#) and [DefaultsSpecification](#)/[DftSpecification](#)) are located. You can access the Core wrappers comprising the Tesson Core Description syntax in the tcd partition. Using the partition names meta: and meta:tcd

allows you to access the metadata configuration elements that define the default and tcd partitions. [Table 5-7](#) summarizes the available partition names.

Table 5-7. Available Partition for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the Default partition is accessed which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tesson Core Description syntax is accessed
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

- `-show_unspecified`

An optional Boolean switch that specifies to include all properties and wrappers of type Repeatable:single, as described in the documentation for the [Wrapper](#) command, in the reported data as opposed to only those that are currently specified. The unspecified properties are shown with their default values.

- `-levels integer / all`

A property that specifies the number of sub-wrapper levels to display. By default, the “all” value is used which specifies to display the full content of all sub-wrappers. When an integer N value is specified, the contents of the sub-wrapper that is N levels below the top wrapper being reported is left empty.

- `-indent left_indent`

An optional switch and integer pair that specifies the indentation to use for the inner wrappers. When this switch is not specified, each inner wrapper is indented two white spaces relative to their parent wrapper. This switch enables you to control the number of white spaces used when indenting a lower wrapper relative to its parent wrapper.

- `-left_indent left_indent`

An optional switch and integer pair that specifies to add a number of white spaces in front of every row in the report. By default, the outer wrapper of the report has zero white spaces and each subsequent inner wrapper adds two white spaces. When `-left_indent` is specified, the outer wrapper has N number of white spaces, as specified by `left_indent`, and each inner wrapper adds two white spaces on top of N. This is useful when you want to report the

contents of a wrapper and insert it inside a “read_config_data -from_string” command as shown in [Example 3](#).

Examples

Example 1

The following example creates some configuration elements and then reports them. The last command uses the “> myfile” to output the reported data into a file called *myfile*.

```
add_config_element tmp(1)
{/tmp(1)}

add_config_element tmp(1)/p1 -value v1 -type property
{/tmp(1)/p1}

add_config_element tmp(1)/p2 -value v2 -type property
{/tmp(1)/p2}

add_config_element tmp(1)/w1 -type wrapper
{/tmp(1)/w1}

add_config_element tmp(1)/w1/P1 -value v1 -type property
{/tmp(1)/w1/P1}

add_config_element tmp(1)/w1/P2 -value v2 -type property
{/tmp(1)/w1/P2}

add_config_element tmp(1)/w2 -copy_from tmp(1)/w1
{/tmp(1)/w2}

report_config_data

tmp(1) {
    p1 : v1;
    p2 : v2;
    w1 {
        P1 : v1;
        P2 : v2;
    }
    w2 {
        P1 : v1;
        P2 : v2;
    }
}
```

report_config_data tmp(1) -level 1

```
tmp(1) {
    p1 : v1;
    p2 : v2;
    w1 {
        // Not shown
    }
    w2 {
        // Not shown
    }
}
```

report_config_data tmp(1) -level 1 > myfile

Example 2

This example shows how config data is reported when it is of type CSV. The `read_config_data` command is used to parse a CSV file generated from Excel. Notice how the format of the data is changed when only reporting the content of the CSV wrapper as opposed to the CSV Wrapper itself. The same feature exists in the `write_config_data` command such that you can export CSV data back into a spreadsheet application.

add_config_element csv(1)

CDV(1)

read_config_data mydata.csv -in csv(1)

report_config_data

```
csv(1) {
    "", "", "", "", "";
    a, b, c;
    "a,c", d;
    "", "", "";
}
```

report_config_data csv(1)

```
'''
a,b,c
"a,c",d
'''
```

Example 3

The following example uses the `-left_indent` switch to report the content of a wrapper and insert it inside a “`read_config_data -from_string`” command. Notice in the second invocation of the `report_config_syntax` command how the `HostScanInterface(ijtag)` wrapper is pushed to the right two white spaces because the command included the `-left_indent` option.

```
ANALYSIS> report_config_data \
    DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag)
```

```

HostScanInterface(ijtag) {
    Sib(sti) {
        Attributes {
            tesson_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
    }
}

ANALYSIS> report_config_data \
DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag) \
-left_indent 2

HostScanInterface(ijtag) {
    Sib(sti) {
        Attributes {
            tesson_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
    }
}

```

You can then easily add the content of the reported config data into the “read_config_file -from_string” wrapper to have it automatically added.

```

ANALYSIS> read_config_file \
-in DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag) \
-from_string

HostScanInterface(ijtag) {
    Sib(sti) {
        Attributes {
            tesson_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
    }
}

```

Related Topics

[report_config_syntax](#)
[write_config_data](#)

report_config_messages

Context: all contexts

Mode: all modes

Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.

Usage

```
report_config_message config_object
    [-hierarchical]
    [-all | [-error] [-warning] [-info]]
```

Description

Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.

Arguments

- ***config_object***

A required value that specifies the name of a configuration element or a collection containing one configuration element.

- **-hierarchical**

An optional switch that can only be used when the config_object_spec is a wrapper and specifies that the command should also return elements below the specified wrapper having at least one config message of the specified types.

- **-all**

An optional Boolean option that specifies that all message types are to be returned. The use of the -all option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the four options is specified, -all is assumed.

- **-error**

An optional Boolean option that specifies that the messages to be returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning, and -info options is specified, -all is assumed.

- **-warning**

An optional Boolean option that specifies that the messages to be returned are of type warning. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning, or -info options are specified, -all is assumed.

- -info

An optional Boolean option that specifies that the messages to be returned are of type info. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning, or -info options are specified, -all is assumed.

Examples

The following example adds configuration messages of type error, warning and info on some elements and reports them.

```
add_config_element tmp(1)
{/tmp(1)}

add_config_element tmp(1)/p1 -value v1 -type property
{/tmp(1)/p1}

add_config_element tmp(1)/p2 -value v2 -type property
{/tmp(1)/p2}

add_config_element tmp(1)/w1 -type wrapper
{/tmp(1)/w1}

add_config_element tmp(1)/w1/P1 -value v1 -type property
{/tmp(1)/w1/P1}

add_config_element tmp(1)/w1/P2 -value v2 -type property
{/tmp(1)/w1/P2}

add_config_element tmp(1)/w2 -copy_from tmp(1)/w1
{/tmp(1)/w2}

add_config_message "Message1" -config_object tmp(1)/w1 -error
{/tmp(1)/w1}

add_config_message "Message2" -config_object tmp(1)/w1/P2 -info
{/tmp(1)/w1/P2}

add_config_message "Message3" -config_object tmp(1)/w1/P2 -warning
{/tmp(1)/w1/P2}

report_conf_message tmp(1)/w1
Report for errors, warnings, infos
/tmp(1)/w1
E0) Message1
```

report_conf_message tmp(1)/w1 -hierarchical

```
Report for errors, warnings, infos
/tmp(1)/w1
  E0) Message1
  /tmp(1)/w1/P2
    W0) Message3
    I0) Message2
```

Related Topics

[add_config_message](#)
[get_config_messages](#)

report_config_syntax

Context: unspecified, all contexts

Mode: all modes

Reports the legal configuration syntax for a specified configuration object.

Usage

```
report_config_syntax config_or_meta_object_spec
    [-properties_only | -wrappers_only]
    [-levels integer]
```

Description

Reports the legal configuration syntax found below the specified config_or_meta_object_spec element.

If you refer to the meta path, you only need to specify the wrapper and property names such as DftSpecification/IjtagNetwork even though the real path name is Wrapper(DftSpecification)/Wrapper(IjtagNetwork). If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it will look up the metadata of the element and report its syntax.

Three options limit the amount of syntax displayed to only wrapper, only properties, or to a specified sub-level count.

Arguments

- **config_or_meta_object_spec**

A required string that specifies the configuration wrapper for which you want to see the available syntax. If you refer to the meta path, you only need to specify the wrapper and property names such as “DftSpecification/IjtagNetwork” even though the real meta path name is “Wrapper(DftSpecification)/Wrapper(IjtagNetwork)”. If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it will look up the metadata of the element and report its syntax.

- -properties_only

An optional Boolean option that restricts the command to only show the properties at the top level of the specified wrapper.

- -wrappers_only

An optional Boolean option that restricts the command to only show the wrappers and exclude the properties.

- -level *integer*

An optional value pair that specifies the number of sub-wrappers to show. When unspecified, all sub-levels are shown.

Examples

Example 1

The following example uses the report_config_syntax command to show the available syntax, limited to two levels, currently available in the DftSpecification wrapper.

```
set_context dft -rtl
report_config_syntax DftSpecification -level 2

DftSpecification(<module_name>,<id>) {
    rtl_extension           : <string> ; // default: v
    gate_extension          : <string> ; // default: vg
    reuse_modules_when_possible : <boolean> ; // default: auto
    IJtagNetwork {
        DataInPorts {
        }
        DataOutPorts {
        }
        HostScanInterface(<id>) {
        }
    }
}
```

Example 2

The second example shows the entire syntax available in the Tdr wrapper.

```
report_config_syntax \
DftSpecification/IJtagNetwork/HostScanInterface/Tdr
```

```
DftSpecification(<module_name>,<id>) {
    IjtagNetwork {
        HostScanInterface(<id>) {
            Tdr(<id>) {
                Interface {
                    tck          : <port_name>           ; // default: ijtag_tck
                    reset       : <port_name>           ; // default: ijtag_reset
                    select      : <port_name>           ; // default: ijtag_sel
                    shift_en    : <port_name>           ; // default: ijtag_se
                    capture_en  : <port_name>           ; // default: ijtag_ce
                    update_en   : <port_name>           ; // default: ijtag_ue
                    scan_in     : <port_name>           ; // default: ijtag_si
                    scan_out    : <port_name>           ; // default: ijtag_so

                    reset_polarity : <active_polarity> ; // legal : active_high
                    (active_low)
                }
                DataInPorts {
                    count      : <int>           ; // default: auto
                    port_naming : <port_name>, ... ; // default: ijtag_data_in[%d]
                    connection(<range>) : <pin_name> ; // repeatable
                }
                DataOutPorts {
                    count      : <int>           ; // default: auto
                    port_naming : <port_name>, ... ; // default: ijtag_data_out[%d]
                    multiplexing : <boolean>         ; // default: auto
                    output_timing(<range>) : <timing_spec> ; // repeatable
                    connection(<range>) : <pin_name> ; // repeatable
                }
                parent_instance          : <instance_name> ;
                leaf_instance_name       : <leaf_instance_name> ;
                keep_active_during_scan_test : <boolean>           ; // default: auto
                length                 : <int>           ; // default: auto
                extra_bits_capture_value : <enum>           ; // legal : 1 (0) self
                reset_value             : <binary>          ; // default: auto
                Attributes {
                    <attribute> : <string> ; // repeatable
                }
                DecodedSignal(<signal_name>) {
                    decode_values : <binary_code>, ... ;
                    multiplexing  : <boolean>           ; // default: auto
                    output_timing : <timing_spec> ; // legal : (auto) unlatched normal
                    ijtag_scan_selection
                    connection : <port_pin_name> ; // repeatable
                    Attributes {
                        <attribute> : <string> ; // repeatable
                    }
                }
            }
        }
    }
}
```

Related Topics

[create_dft_specification](#)

[report_config_data](#)

report_context

Context: unspecified, all contexts

Mode: all modes

Reports the current context as specified by the set_context command and any inferred subcontexts.

Usage

```
report_context
```

Description

Reports the current context as specified by the set_context command and any inferred subcontexts.

This command also provides information about inferred sub-contexts as shown in the examples below.

Arguments

None

Return Values

None

Examples

Example 1

In the following example, the command reports that it ran in uncompressed ATPG mode.

```
report_context
// Context: patterns -scan -no_rtl
// Uncompressed ATPG (no EDT) mode active.
```

Example 2

In the following example, the command reports that it is in extraction mode.

```
report_context
// Context: patterns -ijtag -no_rtl
// ICL extraction mode active.
```

Example 3

In the following example, the command reports that it is in compressed ATPG mode.

```
report_context
// Context: patterns -scan -no_rtl
// Compressed ATPG (EDT) mode active.
```

Example 4

In the following example, the command reports that it is in Logic BIST mode.

```
report_context
// Context: patterns -scan -no_rtl
// Logic BIST mode active.
```

Related Topics

[get_context](#)

[set_context](#)

report_control_signals

Context: dft -scan

Mode: analysis

Displays the rules checking results for control signals consisting of clocks added with the `add_clocks` command and pins identified for gating scannable memory elements with test logic.

Usage

```
report_control_signals {[-All] | {[pin_pathname...]} [-Clock] [-Set] [-Reset] [-Write] [-Read]  
[-Tristate_enable]} [-Verbose] [-NOSTABLE_High] [-NOSTABLE_Low]
```

Description

Displays the rules checking results for control signals consisting of clocks added with the `add_clocks` command and pins identified for gating scannable memory elements with test logic.

Arguments

- **-All**
An optional switch that outputs all control signals to the report. Default setting.
- ***pin_pathname***
An optional, repeatable string that specifies the pathnames of control signals to report on.
- **-Clock**
An optional switch that outputs clock control signals in the report.
- **-Set**
An optional switch that reports on all set control signals.
- **-Reset**
An optional switch that reports on all reset control signals.
- **-Write**
An optional switch that reports on all write control signals.
- **-Read**
An optional switch that reports on all read control signals.
- **-Tristate_enable**
An optional switch that reports on all tristate enable signals.
- **-Verbose**
An optional switch that includes all memory elements associated with each control signal in the report.
- **-NOSTABLE_High**
An optional switch that disables the report on any stable-high memory elements.

- -NOSTABLE_Low

An optional switch that disables the report on any stable-low memory elements.

Related Topics

[add_clocks](#)

[delete_clocks](#)

report_core_descriptions

Context: all contexts
Mode: all modes
Reports all core description files.

Usage

```
report_core_descriptions [-all]
```

Description

Reports all core description files.

The tool uses the information in this file to retarget patterns for instances of the associated core. Since it is useful have know the *design_id* for cores, the report includes the *design_id* value taken from memory or the TSDB container. The core description report is sorted by core name, then the *design_id* value and finally by mode name.

The *design_id* for TCD instruments are not stored in memory, but is included in the instrument's name.

Arguments

- **-all**
An optional literal that reports the core descriptions in the open [Tessent Shell Data Base \(TSDB\)](#) as well as the core descriptions in memory.

Examples

The following example shows the output of the report_core_descriptions command in two modes:

Example 1

This command in setup mode:

```
report_core_descriptions
Name          Mode Name      Type
-----        -----
* chip        retargeting   retargeting
Note: * denotes current design
```

Example 2

This command in analysis mode:

```
report_core_descriptions
```

Command Dictionary (R) report_core_descriptions

Name	Mode Name	Type	Instantiations	Chain Patterns	Scan Patterns
* chip	retargeting	retargeting		8	117
piccpu_maxlen16	internal	internal	1		

Note: * denotes current design

Example 3

In this example, there are two cores inside the opened TSDBs: coreA and coreB. Every core has multiple views specified with mode and design_id values. The report sorts the cores by core name, then by *design_id* value, and finally by the mode name.

Name	Design ID	Mode Name	Type	In Memory	TCD File
* top		unwrapped	unwrapped	Yes	
coreA	id1	m1	internal	No	tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA_atpg_mode_m1/coreA_m1.tcd.gz
	id1	m2	internal	No	tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA_atpg_mode_m2/coreA_m2.tcd.gz
	id1	m3	external	No	tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA_atpg_mode_m3/coreA_m3.tcd.gz
	id2	m1	internal	No	tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA_atpg_mode_m1/coreA_m1.tcd.gz
	id2	m2	internal	No	tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA_atpg_mode_m2/coreA_m2.tcd.gz
	id2	m3	external	No	tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA_atpg_mode_m3/coreA_m3.tcd.gz
coreA_id1_tessent_edt_blk1		edt	No		tsdbA/instruments/coreA_id1_edt.instrument/coreA_id1_tessent_edt_blk1.tcd
coreA_id1_tessent_edt_blk2		edt	No		tsdbA/instruments/coreA_id1_edt.instrument/coreA_id1_tessent_edt_blk2.tcd
coreB	rt11	bcd	internal	No	tsdbB/logic_test_cores/coreB_rt11.logic_test_core/coreB_atpg_mode_bcd/coreB_bcd.tcd.gz
	rt12	bcd	internal	No	tsdbB/logic_test_cores/coreB_rt12.logic_test_core/coreB_atpg_mode_bcd/coreB_bcd.tcd.gz
coreB_id2_tessent_edt		edt	No		tsdbB/instruments/coreB_id2_edt.instrument/coreB_id2_tessent_edt.tcd

Note: * denotes current design

Example 4

In this example, a number of TCDs have been loaded into memory for coreA and coreB. Every mode has an associated *design_id* value.

report_core_descriptions

Name	Design ID	Mode Name	Type
* top		unwrapped	unwrapped
coreA	id1	m1	internal
	id1	m2	internal
	id2	m3	external
coreA_id1_tessent_edt_blk1		edt	
coreA_id2_tessent_edt_blk1		edt	
coreB	rt12	bcd	internal
coreB_id2_tessent_edt		edt	

Note: * denotes current design

Related Topics

- [add_core_instances](#)
- [delete_core_instances](#)
- [report_core_instances](#)

[**write_core_description**](#)

report_core_instance_parameters

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: all modes

Reports the instrument configuration parameters and their values that were specified when the core instance was added with the add_core_instances commands -parameter_values switch.

Usage

```
report_core_instance_parameters {-cores core_names | -modules module_objects |  
-instances instance_objects} [-silent]
```

Description

Reports the EDT instrument configuration parameters and their values that were specified when the core instance was added with the add_core_instances commands -parameter_values switch. By default, the command reports all parameters when you have no filters defined.

Arguments

- **-cores *core_names***

A required switch and Tcl list that specify the core names of the core descriptions from which all instrument configuration parameters should be reported.

- **-modules *module_objects***

A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance instrument configuration parameters for each instantiation of those modules in the design.

- **-instances *instance_objects***

A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance EDT instrument configuration parameters for each instance in the list.

- **-silent**

An optional switch that specifies to suppress warning messages.

Examples

```
SETUP> report_core_instance_parameters -core  
{m8051_edt m8051_G_tessent_occ_NX1 m8051_G_tessent_occ_NX2}
```

Core instance 'm8051_edt_i' (core 'm8051_edt', instrument type 'edt')

Parameter Name	Parameter Value	Legal Override Values	Read Only
edt_bypass	off	off, on	
edt_single_bypass_chain	off	off, on	
tessent_chain_masking			Yes
tessent_chain_masking	off	off, on	

Related Topics

[add_core_instances](#)

[set_core_instance_parameters](#)

report_core_instances

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.

Usage

```
report_core_instances [-cores core_names | -modules module_objects |  
-instances instance_objects] [-silent]
```

Description

Reports the instance bindings created with the [add_core_instances](#) command, as well as the pattern count for patterns read in for each core. If this command is invoked in analysis mode, the pattern count for each core is also reported.

You can specify the absolute instance path names or the module or core names. You can either report all core instances that have been added (the default), or filter by modules, instances, or core names.

This command also reports the *design_id* values if they are available. TCD files incoming from TSDBs that have logic_test_core containers created with an empty *design_id* will be reported with an empty value. Since it is useful to know the *design_id* for cores, the report includes the *design_id* value taken from memory or the TSDB container. The core description report is sorted by core name, then the *design_id* value and finally by mode name.

Arguments

- **-cores *core_names***

A switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be reported.

- **-modules *module_objects***

An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each instantiation of those modules in the design.

- **-instances *instance_objects***

An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each instance in the list.

- **-silent**

An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example reports all existing core instances while in setup mode.

report_core_instances

Instance	Core	Mode Name	Type	Module
core_1	my_core_1	internal	internal	my_module_1
core_2	my_core_2	internal	internal	my_module_2
core_3	my_core_3	internal	internal	my_module_3

Example 2

The following example reports all existing core instances while in analysis mode.

report_core_instances

Instance	Core	Mode Name	Type	Module	Chain Patterns	Scan Patterns
core_1	my_core_1	internal	internal	my_module_1	12	10
core_2	my_core_2	internal	internal	my_module_2	12	10
core_3	my_core_3	internal	internal	my_module_3	6	74

Example 3

The following example first reports all core instances for modules my_module_1 and my_module_2 while in setup mode; it then reports the same while in analysis mode.

report_core_instances -modules {my_module_1 my_module_2}

Instance	Core	Mode Name	Type	Module
core_1	my_core_1	internal	internal	my_module_1
core_2	my_core_2	internal	internal	my_module_2

set_system_mode analysis

...

report_core_instances -modules {my_module_1 my_module_2}

Instance	Core	Mode Name	Type	Module	Chain Patterns	Scan Patterns
core_1	my_core_1	internal	internal	my_module_1	12	10
core_2	my_core_2	internal	internal	my_module_2	12	10

Example 4

The following example reports all EDT IP instances:

report_core_instances

Instance	Core	Type	Module
blk1_edt_i	blk1_edt	edt	blk1_edt
blk2_edt_i	blk2_edt	edt	blk2_edt
blk3_edt_i	blk3_edt	edt	blk3_edt
blk4_edt_i	blk4_edt	edt	blk4_edt

Example 4

The following example first reports core instances core_1 and core_3 while in setup mode; it then reports the same while in analysis mode.

```
report_core_instances -instances {core_1 core_3}
```

Instance	Core	Mode	Name	Type	Module
core_1	my_core_1	internal	internal	internal	my_module_1
core_3	my_core_3	internal	internal	internal	my_module_3

```
set_system_mode analysis
```

```
...
```

```
report_core_instances -instances {core_1 core_3}
```

Instance	Core	Mode	Name	Type	Module	Chain Patterns	Scan Patterns
core_1	my_core_1	internal	internal	internal	my_module_1	12	10
core_3	my_core_3	internal	internal	internal	my_module_3	6	74

Example 5

In this example, the tool reports the core instances with the *design_id* values.

```
SETUP> report_core_instances
Instance          Core           Design ID Mode Name Type   Module
-----          -----
'coreA_inst1'    coreA          id1      m1    internal 'coreA'
|_ 'coreA_inst1/coreA_id1_tessent_edt_blk1_i' coreA_id1_tessent_edt_blk1
|_ 'coreA_inst1/coreA_id1_tessent_edt_blk2_i' coreA_id1_tessent_edt_blk2
'coreA_inst2'    coreA          id1      m1    internal 'coreA'
|_ 'coreA_inst2/coreA_id1_tessent_edt_blk1_i' coreA_id1_tessent_edt_blk1
|_ 'coreA_inst2/coreA_id1_tessent_edt_blk2_i' coreA_id1_tessent_edt_blk2
'coreB_inst1'    coreB          rtl2     bcd   internal 'coreB'
|_ 'coreB_inst1/coreB_id2_tessent_edt_i'       coreB_id2_tessent_edt
```

Related Topics

[add_core_instances](#)

[delete_core_instances](#)

[read_core_descriptions](#)

[write_core_description](#)

report_core_parameters

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: all modes

Reports all core parameters and their values.

Usage

```
report_core_parameters {-cores core_names | -modules module_objects |  
-instances instance_objects} [-silent]
```

Description

Reports all core parameters and their values. By default, the command reports all parameters when you have no filters defined.

Arguments

- **-cores *core_names***

A required switch and Tcl list that reports core parameters for the specified *core_names*.

- **-modules *module_objects***

A required switch and string pair that reports core parameters for the specified *module_objects*, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-instances *instance_objects***

A required switch and string pair that reports core parameters for the specified *instance_objects*, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-silent**

An optional switch that specifies to suppress warning messages.

Examples

```
SETUP> rep_core_param -cores piccpu_maxlen16_1_edt  
Core 'piccpu_maxlen16_1_edt' (instrument type 'edt')  
-----  
Parameter Name          Default Value  Legal Values  
-----  
edt_bypass              off           off, on  
edt_configuration        high, low  
edt_low_power_shift_en   on            off, on  
used_input_channels      4             2..4
```

Related Topics

[add_core_instances](#)

[set_core_instance_parameters](#)

report_design_sources

Context: all contexts

Mode: all modes

Reports the pathnames or file extensions previously specified with the set_design_sources command.

Usage

```
report_design_sources
```

Description

Reports the pathnames or file extensions previously specified with the set_design_sources command.

Arguments

None.

Examples

Example 1

The following example shows a report generated by this command:

```
set_design_sources -format icl -y ..\data\icl_primitives -extension icl
set_design_sources -y ..\data\verilog_primitives -extension v
report_design_sources

// ICL search_design_load_path: activated
// -----
// format      type    path                                file extensions
// -----      ----
// Verilog     dir     '..\data\verilog_primitives'   'v v.gz'
// ICL         dir     '..\data\icl_primitives'       'icl icl.gz'
```

Example 2

The following example reports information about all formats:

```
SETUP> set_design_sources -format tcd_scan -y ..\data -extension tcd_scan
SETUP> report_design_sources
```

```
// Search design load path settings
// =====
// format          value
// -----  -----
// icl            on
// tcd_bscan      on
// tcd_scan       on
// tcd_fusebox    on
// tcd_memory     on
// tcd_memory_cluster on
//
// File and directory search path
// =====
// format          type   path      file extensions
// -----  -----  -----
// verilog         dir    ./.data   v vhd v.gz vhd.gz
// tcd_scan        dir    ..../data tcd_scan tcd_scan.gz
// tcd_fusebox     dir    ./.data   tcd_fbox tcd_fbox.gz
// tcd_memory      dir    ./.data   tcd_mem_lib tcd_mem_lib.gz
// tcd_memory_cluster dir   ./.data   tcd_mem_cluster_lib
//                                         tcd_mem_cluster_lib.gz
//
```

Related Topics

[get_design_sources](#)
[set_simulation_library_sources](#)
[set_design_sources](#)

report_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Returns a tabular report of the DFM rules contained in the layout database.

Usage

```
report_dfm_rules [-header | -col columns...]
```

Description

Returns a tabular report of the DFM rules contained in the layout database.

By default, all columns are listed. These are: id, rule, layer, and type.

For more information about DFM rules, see “[Diagnosis for Design for Manufacturability Analysis](#).”

Arguments

- **-header**
An optional switch that returns a list of available column names.
- **-col *columns...***
An optional switch that specifies the column names you want to use for the DFM rule information in the report.

Examples

Example1

The following example creates a report that contains all available columns.

```
report_dfm_rules
```

id	rule	layer	type
1	minimum_space_4_line_route1	route_1	BRIDGE
2	metal_cross_edge_route1	route_1	OPEN
3	low_density_min_space_route2	route_2	BRIDGE
4	half_comb_route3	route_2	OPEN
5	metal_cross_edge_route1	route_3	OPEN
6	corner_to_interior_metal4	route_4	BRIDGE
7	divergent_line_with_jog_metal5	route_5	BRIDGE
8	metal_cross_edge_route1	route_5	OPEN
9	route_5_barbell	route_5	OPEN
10	route_2_between_vias	route_2	OPEN

Example 2

The following example creates a report that contains the rule name and the type.

report_dfm_rules -col rule type

rule	type
minimum_space_4_line_route1	BRIDGE
metal_cross_edge_route1	OPEN
low_density_min_space_route2	BRIDGE
half_comb_route3	OPEN
metal_cross_edge_route1	OPEN
corner_to_interior_metal4	BRIDGE
divergent_line_with_jog_metals5	BRIDGE
metal_cross_edge_route1	OPEN
route_5_barbell	OPEN
route_2_between_vias	OPEN

Related Topics

[get_dfm_rules](#)[delete_dfm](#)[import_dfm](#)

report_dft_clock_enables

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the `set_current_design` command.

Reports the currently defined DFT clock enables.

Usage

```
report_dft_clock_enables [-post_drc] [-usage func_en | test_en]
```

Description

Reports the currently defined DFT clock enables.

A DFT clock enable signal is either automatically learned on the ports of clock gating cells, or is specified using the `add_dft_clock_enable` command on a pin or port of the design.

The `-post_drc` switch can be specified in analysis mode, when the `-usage` switch defaults to or is specified as “`func_en`”, to report the `func_en` pin that will be intercepted to enable the clock gaters that are in the fanin of the memory clocks during memory BIST testing.

Arguments

- `-usage func_en | test_en`

An optional switch and value pair that specifies the usage of the clock enable pins and ports to report. When you specify the “`func_en`” value, the report includes the ports on clock gating cells that have the *function* attribute set to “`func_enable`”. The report also includes the ports or pins defined with the `add_dft_clock_enables` -usage `func_en` option. When you specify the “`test_en`” value, the report includes the ports on clock gating cells that have the *function* attribute set to “`test_enable`”. The report also includes the ports or pins defined with the `add_dft_clock_enables` -usage `test_en` option.

- `-post_drc`

An optional Boolean switch that can be specified in analysis mode, when the `-usage` switch defaults to or is specified as “`func_en`”, to report the `func_en` pin that will be intercepted to enable the clock gaters that are in the fanin of the memory clocks during memory BIST testing. Without this switch, the command reports the clock enable ports or pins as defined or learned in the cell library or as defined by the “`add_dft_clock_enables` -usage `func_en`” command. When the `-post_drc` switch is specified, the ports are mapped to pins and only those fanning out to memory clock pins are reported.

Examples

The following example uses the `add_dft_clock_enables` command to identify the en ports of clock gating modules whose module name starts with “`my_clock_gate`”, and to define the pin on an instance as a clock enable signal with a polarity of 0. The source of the clock enable signal is a special TDR that is scan tested with the rest of the functional logic. The

[add_dft_control_points](#) command is used to control the select signal of an existing clock multiplexer; the [add_dft_clock_enable](#) command is not used to control that signal because the clock multiplexer must be controlled in scan test mode too.

The [report_dft_clock_enables](#) command is used in analysis mode with and without the [-post_drc](#) option to show the specified clock enables and those extracted during DRC. The specified clock enables can exist on pins or ports of the current design or on ports of any sub-module. During DRC, the clock enables specified on the ports of sub-modules are mapped to pins. Only the clock enable signals that affect the memory clocking are kept.

```
set_context dft -rtl
# read design command here
set_current_design my_design
set_design_level physical_block
set_dft_specification_requirements -memory_test on

add_clocks clka -period 6.7ns

set mods [get_modules my_clock_gate*]
add_dft_clock_enable [get_ports en -of_modules $mods] -usage func_en
add_dft_clock_enable clock_ctrl/disable -usage func_en -active_polarity 0

add_dft_control_point clock_mux/select

check_design_rules
report_dft_clock_enable

// Dft clock enable for usage 'func_en'
// -----
// -----          polarity
// Node           polarity
// -----
// Port 'en' of 'my_clock_gate_and'      1
// Port 'en' of 'my_clock_gate_or'       1
// Pin  '/clock_ctrl/disable'            0

report_dft_clock_enable -post_drc

// Dft clock enable for usage 'func_en'
// -----
// -----          polarity
// Node           polarity
// -----
// '/clock_ctrl/master_clock_gate/en'    1
// '/uart/mem_clk_gate1/en'              1
// '/clock_ctrl/disable'                 0
```

Related Topics

[add_dft_clock_enables](#)
[delete_dft_clock_enables](#)
[read_cell_library](#)

report_dft_clock_muxes

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the `set_current_design` command.

Reports the currently defined DFT clock muxes that were previously added using the `add_dft_clock_muxes` command.

Usage

```
report_dft_clock_muxes
```

Description

Reports the currently defined DFT clock muxes that were previously added using the [add_dft_clock_mux](#) command.

Arguments

None

Examples

The following example adds three DFT clock muxes and then reports them.

```

add_dft_clock_muxes clkb -test_clock_source mem1/clk -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_clock_muxes pll1/vco -test_clock_source pll1/ref \
    -dft_signal_source_name pll_bypass
add_dft_clock_muxes pll1/ref -test_clock_source clkb -dft_signal_source_name alt_ref
report_dft_clock_muxes

// Dft clock muxes
// =====
// ----- -----
//   Node      Test clock source  Control source name
// ----- -----
// 'clkb'     mem1/clk          all_test
// 'pll1/vco' pll1/ref          pll_bypass
// 'pll1/ref' clkb              alt_ref
// 
```

Related Topics

[add_dft_clock_mux](#)[delete_dft_clock_muxes](#)

report_dft_control_points

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the `set_current_design` command.

Reports the currently defined DFT control points that were previously added using the `add_dft_control_points` command.

Usage

```
report_dft_control_points
```

Description

Reports the currently defined DFT control points that were previously added using the `add_dft_control_points` command.

Arguments

None

Examples

Example 1

The following example adds three DFT control points and then reports them.

```
add_dft_control_point portb -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_control_point pll1_mux/s -dft_signal_source_name pll_bypass
add_dft_control_point ref_mux/s -dft_signal_source_name alt_ref
report_dft_control_points

// DFT Control points
// =====
// -----
//     Node      Control source name
// -----
// 'portb'    all_test
// 'pll1_mux/s' pll_bypass
// 'ref_mux/s' alt_ref
//
```

Example 2

If you issue the `report_dft_control_points` command in analysis mode, the report shows if the added `add_dft_control_points`-type AsyncSetReset nodes are ignored or reached by the [DFT_C9](#) DRC as follows:

The report shows the following *before* analysis:

```
// '<node>'    async_set_reset    scan_en    User Added
```

The report shows the following *after* analysis:

```
// '<node>'    async_set_reset    scan_en    User Added, Ignored
```

when the node was not reached by DFT_C9.

When the node was reached by DFT_C9, then the report shows the following:

```
// '<node>'    async_set_reset    scan_en    User Added, Used by DFT_C9,  
Active High
```

-or-

```
// '<node>'    async_set_reset    scan_en    User Added, Used by DFT_C9,  
Active Low
```

The reported polarity is the polarity that activates the reset/set at the flops.

The DFT control logic uses scan_en to force the node inactive during shift mode.

Related Topics

[add_dft_control_points](#)

[delete_dft_control_points](#)

report_dft_modal_connections

A command to report the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands

Usage

```
report_dft_modal_connections
```

Description

This command reports the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands. The report has four optional sections. If called before any DFT modal connections are specified, the command echos:

```
SETUP> report_dft_modal_connections
There are no DFT_modal connections to report
```

Once there are input and/or output DFT modal connections specified, the report includes the “Modes” section which lists all the used modes and the ports/auxiliary data pins used by each mode. It also includes the “Input connections” and/or the “Output connections” sections showing the destination/source of the port or auxiliary data pins in each mode. Finally, if the specification implies the presence of multiplexing at one or more input data destination nodes, the “Input destinations” section is added and the multiplexing at the destination nodes that include multiplexing are shown. See the example below showing the four sections in the report.

Arguments

None

Examples

The following example shows the four sections of the report representing the effect of the used add_dft_modal_connections commands.

```
set mode edt_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes * \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes      * \
    -enable_dft_signal $mode -pipeline_stages 3
```

```

set mode retargeting1_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4

set mode retargeting2_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 5

set mode retargeting3_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[3] \
    -output_data_source_nodes corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4

set mode retargeting4_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[4] \
    -output_data_source_nodes corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4
add_dft_modal_connections -ports gpio[2:3] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[5] \
    -output_data_source_nodes corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 5

SETUP> report_dft_modal_connectionsDft Modal Connections
=====

Modes
-----
edt_mode : DFT signal with usage scan_mode(unwrapped)
Input connections (2):
  Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
  Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
  Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
Input connections (2):
  Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'

```

Command Dictionary (R)
report_dft_modal_connections

```
Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
  Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting2_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connections (2):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
    Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'
retargeting4_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (4):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxIn'
    Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxIn'
  Output connections (2):
    Port 'gpio[4]' through auxiliary output pin 'DEF_inst/gpio_4_AuxOut'
    Port 'gpio[5]' through auxiliary output pin 'DEF_inst/gpio_5_AuxOut'
Input connections
-----
1) Port 'gpio[1]' through auxiliary input pin 'DEF_inst/gpio_1_AuxIn'
  edt_mode : P2 <Existing destination>
  retargeting1_mode : P1 corea_i1/edt_channels_in[1]
  retargeting2_mode : P2 corea_i2/edt_channels_in[1]
  retargeting3_mode : P1 corea_i1/edt_channels_in[1]
  : P1 corea_i2/edt_channels_in[1]
2) Port 'gpio[0]' through auxiliary input pin 'DEF_inst/gpio_0_AuxIn'
  edt_mode : P2 <Existing destination>
  retargeting1_mode : P1 corea_i1/edt_channels_in[0]
  retargeting2_mode : P2 corea_i2/edt_channels_in[0]
  retargeting3_mode : P1 corea_i1/edt_channels_in[0]
  : P1 corea_i2/edt_channels_in[0]
3) Port 'gpio[2]' through auxiliary input pin 'DEF_inst/gpio_2_AuxIn'
  retargeting4_mode : P2 corea_i2/edt_channels_in[1]
4) Port 'gpio[3]' through auxiliary input pin 'DEF_inst/gpio_3_AuxIn'
  retargeting4_mode : P2 corea_i2/edt_channels_in[0]

Output connections
-----
1) Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
  edt_mode : P3 <Existing source>
  retargeting1_mode : P4 corea_i1/edt_channels_out[0]
  retargeting2_mode : P5 corea_i2/edt_channels_out[0]
  retargeting3_mode : P4 corea_i1/edt_channels_out[0] 2) Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'
  retargeting3_mode : P4 corea_i2/edt_channels_out[0]
3) Port 'gpio[4]' through auxiliary output pin 'DEF_inst/gpio_4_AuxOut'
  retargeting4_mode : P4 corea_i1/edt_channels_out[0]
4) Port 'gpio[5]' through auxiliary output pin 'DEF_inst/gpio_5_AuxOut'
  retargeting4_mode : P5 corea_i2/edt_channels_out[0]
```

Input destinations muxing

- 1) Destination input pin 'corea_i1/edt_channels_in[1]'
retargeting1_mode : P1 DEF_inst/gpio_1_AuxIn
retargeting3_mode : P1 DEF_inst/gpio_1_AuxIn
- 2) Destination input pin 'corea_i2/edt_channels_in[1]'
retargeting2_mode : P2 DEF_inst/gpio_1_AuxIn
retargeting3_mode : P1 DEF_inst/gpio_1_AuxIn
retargeting4_mode : P2 DEF_inst/gpio_2_AuxIn
- 3) Destination input pin 'corea_i1/edt_channels_in[0]'
retargeting1_mode : P1 DEF_inst/gpio_0_AuxIn
retargeting3_mode : P1 DEF_inst/gpio_0_AuxIn
- 4) Destination input pin 'corea_i2/edt_channels_in[0]'
retargeting2_mode : P2 DEF_inst/gpio_0_AuxIn
retargeting3_mode : P1 DEF_inst/gpio_0_AuxIn
retargeting4_mode : P2 DEF_inst/gpio_3_AuxIn

report_dft_signal_names

Context: unspecified, all contexts

Mode: all modes

Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration.

Usage

`report_dft_signal_names`

Description

Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration. The report separates the static and dynamic DFT signals into two tables. The table for the static DFT signals includes six columns that show the specific information about each signal. The table for the dynamic DFT signals includes two columns that show the specific information about each signal.

Arguments

None

Examples

The command has no arguments, and creates two tables showing the static and the dynamic DFT signals. The static DFT signals table has six columns, which makes the table wider than this page. For an example of the actual report, issue the `report_dft_signal_names` command in any context. The list of built-in DFT signals shows the complete variation of the report. The actual report generated by the command is not shown here because it is too wide to fit in this narrow page.

`report_dft_signal_names`

Related Topics

[register_static_dft_signal_names](#)

[report_dft_signals](#)

report_dft_signals

Context: dft, patterns

Mode: setup, analysis, insertion

Reports all the DFT signals added or to-be-created in the design.

Usage

```
report_dft_signals
```

Description

Reports all the DFT signals added or existing in the design. The report is a table with five columns.

- The first column is the name of the DFT signal.
- The second column uses either “To be created” or “Existing” based on whether the DFT signal is scheduled to be created or already exists.
- The third column reports one of these values: “User added”, “Imported from DesignInfo”, “ICL attributes”.
 - “User added” means the DFT signal exists because the [add_dft_signals](#) command was issued for the signal in the current insertion pass.
 - “Imported from DesignInfo” means the DFT signal was added on a Port or is a Dynamic DFT signal. During the insertion pass that added the signal, the information was stored in the *.tcd* file located in the [dft_inserted_designs](#) directory inside the [Tessent Shell Data Base \(TSDB\)](#) directory and was automatically re-imported when using [read_design](#) for the next insertion pass.
 - “ICL attributes” means the signal is a static DFT signal defined as an attributed DataOutPort of an ICL module.
- The forth and fifth column report the type and location, respectively.

See “[Example 2](#)” on page 217 of the [add_dft_signals](#) command description to see the attributes that define static DFT signals.

Arguments

None

Examples

The following example shows the report generated when the command is issued after certain DFT signals were defined using the [add_dft_signals](#) command, but prior to issuing the [process_dft_specification](#) command. Rerunning the [report_dft_signals](#) command after issuing the [process_dft_specification](#) command changes the status to “Existing”, and the null locations become the pin on the TDR that sources the DFT signal.

```
add_dft_signals ltest_en int_ltest_en ext_ltest_en
add_dft_signals scan_en test_clock edt_update \
    -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
    -create_from_other_signals
report_dft_signals

// Existing and to be created DFT signals
// =====
// -----  -----
// Name      Status   Origin   Type   Location
// -----  -----
// edt_clock      To be created User added ClockGateAnd here/
//<persistent_cell_prefix>edt_clock
// edt_update      To be created User added Port      my_edt_update
// ext_ltest_en     To be created User added IJTAG register
// int_ltest_en     To be created User added IJTAG register
// ltest_en        To be created User added IJTAG register
// scan_en         To be created User added Port      my_scan_en
// shift_capture_clock To be created User added ClockGateAnd here/
//                                     <persistent_cell_prefix>shift_capture_clock
// test_clock       To be created User added Port      my_test_clock
```

Related Topics

[add_dft_signals](#)
[report_dft_signal_names](#)

report_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Writes the scan diagnosis report to standard I/O.

Usage

```
report_diagnosis [-ENCoded] [-SHORT]
```

Description

Writes the scan diagnosis report to standard I/O.

Arguments

- **-ENCoded**
An optional switch encoding the suspect's pin pathname, net name, and cell name in the diagnosis report.
- **-SHORT**
An optional switch that suppresses the display of layout-aware diagnosis information in the diagnosis report.

Related Topics

[write_diagnosis](#)

report_display_instances

Context: all contexts

Mode: setup, analysis

Shows netlist information for the specified instances displayed in the Flat Schematic, Hierarchical Schematic, or Data window.

Usage

```
report_display_instances {gate_id# | instance_name} ... | -All  
[-Display {FLAt_schematic | HIEarchical_schematic | DAta}] [-Full]
```

Description

Shows netlist information for the specified instances displayed in the Debug, Design, or Data window. The report_display_instances command reports, in the session transcript, netlist information for the specified instances currently displayed in the DFTVisualizer, Flat Schematic, Hierarchical Schematic, or Data window. The default report includes the information listed in the following table:

Table 5-8. Default Netlist Information Reported for Instances

“set_gate_level Design” in effect	“set_gate_level Primitive” in effect
Instance name	Instance name
DFT library model name	Gate identification number
	Gate type

You can display the same information from a tool’s command line using the report_gates command, whether or not DFTVisualizer is open.

Arguments

- ***gate_id#***

A repeatable integer that specifies a gate whose netlist information you are reporting. The value of the *gate_id#* argument is the unique identification number the tool automatically assigns to every gate within the design during the model flattening process.

- ***instance_name***

A repeatable string that specifies the name of an instance whose netlist information you are reporting.

- **-All**

A switch that specifies to report all instances in the current display netlist (including the compacted gates).

- -Display {FLAt_schematic | HIErarchical_schematic | Data}

A switch and literal that specify the window containing the instance(s) you are reporting. The window choices are as follows:

FLAt_schematic — Specifies the Flat Schematic window. This is the invocation default.

HIErarchical_schematic — Specifies the Hierarchical Schematic window.

Data — Specifies the Data window.

- -Full

A switch that specifies to include the following information for each pin on each reported instance:

- Pin name
- Pin type (input or output)
- Simulated pin data (if appropriate)
- Gates to which that pin connects

Examples

The following example analyzes a DRC violation (automatically invoking DFTVisualizer in the process and displaying the part of the netlist related to the violation in the Flat Schematic window):

```
set_gate_level design analyze_drcViolation c3-1
// command: open_visualizer -display flat_schematic
// Note: Gate report now set to clock_cone (clock=/tck).
// Creating Schematic for 5 instances
```

The DFTVisualizer Transcript window shows the following information for the violation:

```
===== (Violation ID = c3-1) =====
// Note: Clock /tck failed rule C3 on input 3 of /reg_enable (76). (C3-1)
// Note: Source of violation: input 3 of /reg_pstate_2_ (70).
// Note: Gate report now set to clock_cone (clock=/tck).
Creating Schematic for 5 instances
```

The next example shows the differences between the default and the detailed report for the gate where clock /tck failed the C3 DRC:

```
report_display_instances /reg_enable
// /reg_enable dffr
report_display_instances /reg_enable -full
```

```
// /reg_enable dffr
//      D      I  (E)  /ix130/Y
//      CLK    I  (C)  /ix473/Y
//      R      I  (-)  /ix475/Y
//      Q      O  (-)  /tri_pin/E
//      QB     O  (-)
```

Related Topics

[add_display_instances](#)
[analyze_drcViolation](#)
[open_visualizer](#)
[report_gates](#)
[set_gate_report](#)

report_drc_rules

Context: all contexts

Mode: all modes

Displays either a summary of DRC violations (fails) or violation occurrence message(s).

Usage

```
report_drc_rules [-Fails_summary | -Summary | rule_id... | rule_id-occurrence#... | -All_fails]
[ {> | >>} file.pathname ]
```

C1 Usage:

```
report_drc_rules C1 [-EXcluded]
```

D5 Usage:

```
report_drc_rules D5
[ { [-TYpe {I0 | I1 | IX | T0 | T1 | TX | TLA}...]
  [-NOType {I0 | I1 | IX | T0 | T1 | TX | TLA}...]
  [-EDge_triggered | -LEvel_sensitive] } | -Summary ]
[ {> | >>} file.pathname ]
```

E5 Usage:

```
report_drc_rules E5 [-TIMing_exceptions]
```

Description

Displays either a summary of DRC violations (fails) or violation occurrence message(s). This command can display a report in one of two formats:

- **Summary Report** — Lists for each reported design rule, one line of data per rule, the current number of DRC violations (fails), the violation handling, and a brief description of the rule. When you request a summary of D5 violations, the report lists the number of D5 violations of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X, TLA).
- **Occurrence Report** — Lists one or more violation occurrence messages that give details of specific DRC violations

Table 5-9 provides a summary of the available information displayed and the arguments you use to obtain them. Refer to the Arguments subsection for complete details about the arguments.

Table 5-9. Available Information Displayed and Arguments

Desired Display	Rules/Occurrences Covered	Argument
Summary report	Design rules that resulted in violations (fails) during DRC	-Fails_summary
	All design rules	-Summary

Table 5-9. Available Information Displayed and Arguments (cont.)

Desired Display	Rules/Occurrences Covered	Argument
Occurrence report	Specific rule, specific occurrence	<i>rule_id-occurrence#</i>
	Specific rule, all occurrences ¹	<i>rule_id</i>
	All occurrences	-All_fails

1. Additional D5-specific arguments enable you to further customize the D5 information display.

You can use the [set_drc_handling](#) command to change the handling of many of the A (RAM), C (clock), D (data), E (extra), K (EDT), T (trace), and W (timing) rules.

For more information about the design rules, refer to “[Design Rule Checking](#)” on page 2639.

Arguments

- **-Fails_Summary**

A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during DRC:

- Rule identification (ID)
- Number of failures of the rule
- Current handling status of the rule
- Brief description of the rule

This is the default.

Note

 This switch does not display anything if there are no rule violations or the tool has not yet performed DRC.

- **-Summary**

A switch that specifies to display a summary report that varies depending on whether you are reporting on all design rules or just on the D5 rule. (The latter reporting is available only in Tessent FastScan and Tessent TestKompress.)

When you report on all rules (report_drc_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during DRC:

- Rule identification (ID)
- Number of failures of the rule
- Current handling status of the rule
- Brief description of the rule

When you report on just the D5 rule (report_drc_rules D5 -Summary) in Tessent FastScan or Tessent TestKompress, the tool displays the number of non-scan memory elements of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X or TLA) that resulted in a D5 violation during DRC. Refer to the description of the -Type switch for the meaning of the types. See also the “[D5](#)” section for detailed information about the D5 rule.

- *rule_id*

A repeatable string that specifies the identification literal (ID) of a particular design rule for which you want to display the violations. For information about the design rules and their ID, refer to the appropriate section:

- [RAM Rules \(A Rules\)](#)
- [BIST Rules \(B Rules\)](#)
- [Clock Rules \(C Rules\)](#)
- [Scan Cell Data Rules \(D Rules\)](#)
- [Extra Rules \(E Rules\)](#)
- [EDT Finder Rules \(F Rules\)](#)
- [Flattening Rules \(FN, FP, and FG Rules\)](#)
- [General Rules \(G Rules\)](#)
- [EDT Rules \(K Rules\)](#)
- [Procedure Rules \(P Rules\)](#)
- [Scannability Rules \(S Rules\)](#)
- [Scan Chain Trace Rules \(T Rules\)](#)
- [Power-Aware Rules \(V Rules\)](#)
- [Timing Rules \(W Rules\)](#)
- [ICL Extraction Rules \(I Rules\)](#)
- [ICL Semantic Rules \(ICL Rules\)](#)
- [Core Mapping for ATPG and Scan Pattern Retargeting Rules \(R Rules\)](#)

- *rule_id-occurrence#*

A repeatable string that specifies the identification literal (ID) of a particular design rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. For example, you can analyze the second violation occurrence of the C3 rule by specifying C3-2. Numbers to occurrences of rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.

- **-All_Fails**

A switch that specifies to display all occurrence messages for all occurrences of rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a “`report_drc_rules <rule_id>`” command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.

- **>file.pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of `file.pathname`.

- **>>file.pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of `file.pathname`.

C1-Only Arguments

- **C1**

A required literal that specifies reporting C1 DRC rule violations.

- **-EXcluded**

An optional switch for use only with a C1 violation. Specifying this switch reports the C1 violations that have been excluded from the default C1 list because these C1 violations can be handled by the tool without causing potential mismatch.

D5-Only Arguments

- **-TYpe I0 | I1 |IX | T0 | T1 | TX | TLA**

An optional switch and repeatable literal that displays D5 occurrence messages for only the specified type(s) of non-scan sequential elements. The literal choices for the type of element are as follows (the term you will see in occurrence messages for each type is shown in parentheses):

I0 — If the element is at 0 at the beginning of the first capture cycle and may go to any state during capture. (INIT-0)

I1 — If the element is at 1 at the beginning of the first capture cycle and may go to any state during capture. (INIT-1)

IX — If the element’s state is unknown at the beginning of the first capture cycle and may go to any state during capture. (INIT-X)

T0 — If the element is always at 0 during capture. (TIE-0)

T1 — If the element is always at 1 during capture. (TIE-1)

TX — If the element is always at an unknown state during capture. (TIE-X)

TLA — If the element is always transparent when its clock is at its off state. (TLA)

Tip

 Except for TLAs, you can also direct the tool to display information for only those D5 elements that are edge-triggered or level-sensitive. See the `-Edge_triggered` and `-Level_sensitive` switch descriptions for details.

- `-NOType I0 | I1 |IX | T0 | T1 | TX | TLA`

An optional switch and repeatable literal that specify not to display occurrence messages for the particular type(s) of D5 violations. See the description of the `-Type` switch for the meaning of the literal choices.

- `-EDge_triggered | -LEvel_sensitive`

Optional switches that specify to display D5 occurrence messages either for edge-triggered or level-sensitive elements only. The default (when neither option is specified) is to display information for both edge-triggered and level-sensitive elements.

E5-Only Arguments

- `-TIMing_exceptions`

An optional switch that specifies to report only on E5 violations due to timing exceptions. In order to use this switch you must first issue a “`set_drc_handling -timing_exception_check ON`” command in setup mode (see “[Example 8](#)” on page 1491).

Examples**Example 1**

The following example displays a summary of the rules that resulted in violations during an attempted ATPG run:

report_drc_rules

```
C3: #fails=2 handling=warning (clock may capture data affected by its
     captured data)
C5: #fails=1 handling=error (clock is connected to multiple ports of same
     latch)
C7: #fails=19 handling=warning (scan cell capture ability check)
C8: #fails=2 handling=warning (PO connected to a clock line)
C9: #fails=2 handling=warning (PO connected to a clock line gated by scan
     cell that uses same clock)
D5: #fails=23 handling=warning (non-scan memory element)
D6: #fails=22 handling=warning (non-transparent non-scan latches)
D7: #fails=22 handling=warning (stable high edge-triggered clock ports)
```

Example 2

The following example displays the occurrence message for the two C3 violations, and the first C7, and first D7 violation reported in the preceding example:

report_drc_rules c3 c7-1 d7-1

```
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_b_i0.latch  
    (1501). (C3-1)  
// Source of violation: input 3 of /sp_c_i1.slave (1482).  
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_c_i0.latch  
    (1527). (C3-2)  
// Source of violation: input 3 of /sp_c_i1.slave (1482).  
// Warning: Clock input 5 of /c8.master (1495) cannot capture data  
    with single clock on. (C7-1)  
// Warning: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```

Example 3

The following example changes the handling of data rule 7 (D7) from warning to error and also specifies execution of a full test generation analysis when performing the rules checking for the clock (C) rules:

```
set_drc_handling d7 error atpg_analysis  
set_system_mode analysis  
  
//-----  
//Begin scan chain identification process, memory elements=8.  
//-----  
// Reading group test procedure file /user/design/tpf.  
// Simulating load/unload procedure in g1 test procedure file.  
// Chain = c1 successfully traced with scan_cells = 8.  
// Error: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)  
// Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

Example 4

The following example displays the occurrence message for the D7 violation:

```
report_drc_rules d7-1  
//Error: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```

Example 5

The following example displays a summary of the non-scan memory elements that resulted in D5 rule violations:

```
report_drc_rules d5 -summary  
  
// -----  
// 44 non-scan memory elements are identified.  
// -----  
//      9 non-scan memory elements are identified as TIE-0. (D5)  
//      2 non-scan memory elements are identified as TIE-1. (D5)  
//      5 non-scan memory elements are identified as TIE-X. (D5)  
//      3 non-scan memory elements are identified as INIT-0. (D5)  
//     11 non-scan memory elements are identified as INIT-1. (D5)  
//      4 non-scan memory elements are identified as INIT-X. (D5)  
//     10 non-scan memory elements are identified as TLA. (D5)  
// -----
```

Example 6

The following example displays details about the preceding example's D5 violations that occurred on non-scan memory elements the tool identified as type INIT-X:

report_drc_rules d5 -type ix

```
// Warning: /U1 (78) is a non-scan flip-flop identified as INIT-x. (D5-1)
// Warning: /U6 (56) is a non-scan flip-flop identified as INIT-x. (D5-7)
// Warning: /U7 (82) is a non-scan flip-flop identified as INIT-x. (D5-8)
// Warning: /U9 (11) is a non-scan latch identified as INIT-x. (D5-9)
// Number of non-scan memory elements reported = 4.
```

Example 7

The following example displays the information for just the INIT-X non-scan latch:

report_drc_rules d5 -type ix -level_sensitive

```
// Warning: /U9 (11) is a non-scan latch identified as INIT-x. (D5-9)
// Number of non-scan memory elements reported = 1.
```

You could obtain the same information using the -Notype switch:

report_drc_rules d5 -notype i1 i0 tx t1 t0 tla -level_sensitive

```
// Warning: /U9 (11) is a non-scan latch identified as INIT-x. (D5-9)
// Number of non-scan memory elements reported = 1.
```

Example 8

The following example reads timing exception paths and enables the timing exception check for E5. Next, the example enters analysis mode and reports E5 due to timing exceptions.

```
SETUP> read_sdc my_design.sdc
SETUP> set_drc_handling E5 -timing_exception_check on
SETUP> set_system_mode analysis

...
// 9 gates may have an observable X-state, 2 of which are due to timing
// exception paths. (E5)

ANALYSIS> report_drc_rules e5 -timing_exceptions

// DFF gate /M3S004FP/VALIDDEVCHIRPK_REG (59193) may have an observable
// X-state due to timing exception paths. (E5-8)
// DFF gate /M3S014FP0/TXVALIDDH_REG (59220) may have an observable
// X-state due to timing exception paths. (E5-9)
```

Related Topics

[set_drc_handling](#)

report_edt_abort_analysis

Context: dft -edt, patterns -scan (EDT On)

Mode: analysis

Reports the results of EDT Aborted fault analysis.

Usage

```
report_edt_abort_analysis [-block block_name]
    [-from from_cube_id] [-to to_cube_id]
    [-statistics [-cells_most_commonly_specified number_of_cells]
    [-shift_positions_most_commonly_specified number_of_shift_positions]]
    [-cubes]
```

Description

Reports the results of EDT Aborted (EAB) fault analysis. Without any arguments, the report_edt_abort_analysis command generates a summary report containing the cells most commonly specified and the shift positions most commonly specified. The classification of a EAB test cube is meant to identify the EDT block in which the encoding first failed.

See “[EDT Aborted Fault Analysis](#)” in the *Tessent TestKompress User’s Manual* for a complete discussion.

Arguments

- **-block *block_name***

An optional switch and string pair that specifies the name of an EDT block on which to perform the analysis.

- **-from *from_cube_id***

An optional switch and positive integer pair that specifies which EAB test cube the tool should analyze first. The default is 1.

This option is only meaningful when either the -block switch pair is used or all EAB test cubes are from one EDT block. If multiple EDT blocks contain EAB test cubes and -block is not used, then the tool produces an error message.

Used in conjunction with the complementary -to *to_cube_id* switch and positive integer pair.

- **-to *to_cube_id***

An optional switch and positive integer that specifies where to end the EAB analysis. By default, *to_cube_id* is the total number of collected EAB test cubes.

This option is only meaningful when either the -block switch pair is used or all EAB test cubes are from one EDT block. If multiple EDT blocks contain EAB test cubes and -block is not used, then the tool produces an error message.

Used in conjunction with the complementary -from *from_cube_id* switch and positive integer pair.

- -statistics

An optional switch that reports the statistics for the selected block and EAB cubes. This is the default unless you specify the -cubes keyword.

The tool-generated report shows statistics for each EDT block that has EAB cubes. For each such EDT block, the report contains the following sections:

- EAB cube size distribution
- Most specified scan cells by EAB test cubes
- Most specified shift positions by EAB test cubes

Optionally, you can control the number of displayed scan cells and shift positions by using the -cells_most_commonly_specified and -shift_positions_most_commonly_specified options.

- -cells_most_commonly_specified *number_of_cells*

An optional switch and positive integer pair that specifies the number of cells to report. Used in conjunction with the -statistics switch.

- -shift_positions_most_commonly_specified *number_of_shift_positions*

An optional switch and positive integer pair that specifies the number of shift positions to report. Used in conjunction with the -statistics switch.

- -cubes

An optional literal that reports the smallest test cubes for the selected block and test cubes.

Examples

Example 1

The following example shows the default report.

In this example, two EDT blocks contain EAB cubes. The first one contains 3EAB cubes and the second one contains 49. It can be seen from the first one that none of the EAB cubes in the block are “oversized” (>100%).

Note that in this example, the “Gate” column entries are shortened with /*gate_path*, which represents the following:

```
/edt_top_core2_maxlen90/piccpu_maxlen90_i
```

This is done for presentation purposes.

```
ANALYSIS> create_patterns
```

```
.....
```

ANALYSIS> report_edt_abort_analysis

```
//  
//EAB test cube analysis results for EDT block 'piccpu_90':  
//EDT configurations:  
//Number of channels      = 1  
//Number of scan chains   = 5  
//Number of scan cells    = 217  
//Longest scan chain length = 90  
//Approximate encoding capacity = 94  
//  
//EAB test cube size distribution:  
//Specified bits  Specified bits / Capacity (%)  EAB cubes  
//-----  
//28 - 37          30.00% - 40.00%                2  
//37 - 47          40.00% - 50.00%                1  
//Total                         3  
//  
//Analyzed a total of 3 EAB cubes.  
//Top specified scan cells are:  
//ID Specified  Gate                                Chain      Shift  Constraint  
//  times  
//--- -----  
//1 2(66.67%)  '/gate_path/i/phase_reg_0/Q' (31096) piccpu_90_chain1 16  
//2 2(66.67%)  '/gate_path/inst_reg_11/' (31093)  piccpu_90_chain1 13  
//3 2(66.67%)  '/gate_path/inst_reg_10/' (31092)  piccpu_90_chain1 12  
//4 2(66.67%)  '/gate_path/w_reg_2/' (31076)    piccpu_90_chain5 22  
//5 2(66.67%)  '/gate_path/portb_reg_5/' (31006)  piccpu_90_chain5 7  
//6 2(66.67%)  '/gate_path/inst_reg_7/' (31089)  piccpu_90_chain1 9  
//7 2(66.67%)  '/gate_path/inst_reg_6/' (31088)  piccpu_90_chain1 8  
//8 2(66.67%)  '/gate_path/status_reg_6/' (31056) piccpu_90_chain5 17  
//9 2(66.67%)  '/gate_path/inst_reg_3/' (31085)  piccpu_90_chain1 5  
//10 2(66.67%) '/gate_path/inst_reg_2/' (31084)  piccpu_90_chain1 4  
//  
//Top specified shift positions are:  
//Shift position  Specified times  Average specified times per cube  
//-----  
//4           4                  1  
//5           4                  1  
//12          4                  1  
//16          4                  1  
//3           3                  1  
//22          3                  1  
//7           3                  1  
//8           3                  1  
//9           3                  1  
//13          3                  1
```

In the next report block, 8 out of 49 EAB cubes are beyond the encoding capacity. This means that the specified bits of these 8 cubes are over the encoding capacity of the block. It is more or less expected that they cannot be compressed. The other 41 EAB cubes are within normal cube size range.

Note that in this example, the “Gate” column entries are shortened with */gate_path*, which represents the following:

/m8051_5channels/m8051_i

This is done for presentation purposes.

Command Dictionary (R) report_edt_abort_analysis

```
//  
// EAB test cube analysis results for EDT block 'm8051_5ch':  
// EDT configurations:  
// Number of channels = 5  
// Number of scan chains = 48  
// Number of scan cells = 498  
// Longest scan chain length = 11  
// Approximate encoding capacity = 63  
//  
// EAB test cube size distribution:  
// Specified bits Specified bits / Capacity (%) EAB cubes  
// -----  
// 37 - 44 60.00% - 70.00% 1  
// 44 - 50 70.00% - 80.00% 8  
// 50 - 56 80.00% - 90.00% 16  
// 56 - 63 90.00% - 100.00% 16  
// > 63 > 100% 8  
// Total 49  
//  
// Analyzed a total of 49 EAB cubes.  
// Top specified scan cells are:  
//ID Specified Gate Chain Shift Constraint  
// times position  
//---  
//1 29(59.18%) '/gate_path/u14/OPC_reg_4_/' (31458) m8051_5ch_chain26 5  
//2 27(55.10%) '/gate_path/u7/INDIRECT_ADDR_reg_5_/' (31349) m8051_5ch_chain1 4 Condition  
// of NCP  
// user_capture_proc_22  
//3 22(44.90%) '/gate_path/u14/OPC_reg_6_/' (31456) m8051_5ch_chain26 7  
//4 20(40.82%) '/gate_path/u14/OPC_reg_5_/' (31457) m8051_5ch_chain26 6  
//5 20(40.82%) '/gate_path/u14/OPC_reg_1_/' (31461) m8051_5ch_chain26 2  
//6 19(38.78%) '/gate_path/u14/OPC_reg_3_/' (31459) m8051_5ch_chain26 4  
//7 19(38.78%) '/gate_path/u7/INDIRECT_ADDR_reg_3_/' (31351) m8051_5ch_chain1 2 Cell  
// constraint  
//8 18(36.73%) '/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) m8051_5ch_chain1 3 Cell  
// constraint  
//9 18(36.73%) '/gate_path/u1/Q4_reg/' (31164) m8051_5ch_chain23 6  
//10 17(34.69%) '/m8051_5channels/m8051_i/u5/TMPDAT_reg_6_/' (31492) m8051_5ch_chain27 7  
//  
//Top specified shift positions are:  
//Shift position Specified times Average specified times per cube  
//-----  
//7 130 2  
//5 119 2  
//6 108 2  
//4 104 2  
//3 83 1  
//2 62 1  
//8 58 1  
//1 20 0  
//9 10 0  
//0 3 0  
//  
// Total analysis time = 0.09 sec.
```

Example 2

The following example shows a more in depth analysis of the second block in Example 1, which had some incompressible EAB blocks.

Note that in this example, the “Gate” column entries are shortened with */gate_path*, which represents the following:

/m8051_5channels/m8051_i

This is done for presentation purposes.

```
ANALYSIS> report_edt_abort_analysis -block m8051_5ch -from 1 -to 41 -statistics \
-cells_most_commonly_specified 30 -shift_positions_most_commonly_specified 5
```

Command Dictionary (R) report_edt_abort_analysis

```
//  
// EAB test cube analysis results for EDT block 'm8051_5ch':  
// EDT configurations:  
// Number of channels      = 5  
// Number of scan chains   = 48  
// Number of scan cells    = 498  
// Longest scan chain length = 11  
// Approximate encoding capacity = 63  
//  
// EAB test cube size distribution:  
// Specified bits  Specified bits / Capacity (%)  EAB cubes  
// -----  
// 37 - 44          60.00% - 70.00%                1  
// 44 - 50          70.00% - 80.00%                8  
// 50 - 56          80.00% - 90.00%               16  
// 56 - 63          90.00% - 100.00%              16  
// Total           60.00% - 100.00%               41  
//  
// Analyzed a total of 41 EAB cubes.  
// Top specified scan cells are:  
// ID Specified  Gate                                Chain      Shift  Constraint  
// times  
// ---  
// 1 23(56.10%)  '/gate_path/u14/OPC_reg_4_/' (31458) m8051_5ch_chain26 5  
// 2 21(51.22%)  '/gate_path/u7/INDIRECT_ADDR_reg_5_/' (31349) m8051_5ch_chain1 4 Condition  
//                                     of NCP  
//                                     user_capture_proc_22  
// 3 19(46.34%)  '/gate_path/u14/OPC_reg_6_/' (31456) m8051_5ch_chain26 7  
// 4 18(43.90%)  '/gate_path/u7/INDIRECT_ADDR_reg_3_/' (31351) m8051_5ch_chain1 2 Cell  
//                                     constraint  
// 5 18(43.90%)  '/gate_path/u14/OPC_reg_1_/' (31461) m8051_5ch_chain26 2  
// 6 17(41.46%)  '/gate_path/u14/OPC_reg_3_/' (31459) m8051_5ch_chain26 4  
// 7 17(41.46%)  '/gate_path/u14/OPC_reg_5_/' (31457) m8051_5ch_chain26 6  
// 8 16(39.02%)  '/gate_path/u15/u10_DAT_reg_3_/' (31265) m8051_5ch_chain40 5 ATPG  
//                                     constraint on  
//                                     '/gate_path/u15/u10_DAT_reg_3_/Q' (1457)  
// 9 16(39.02%)  '/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) m8051_5ch_chain1 3 Cell  
//                                     constraint  
// 10 15(36.59%) '/gate_path/u5/TMPDAT_reg_6_/' (31492) m8051_5ch_chain27 7  
// 11 14(34.15%) '/gate_path/u1/Q4_reg/' (31164) m8051_5ch_chain23 6  
// 12 14(34.15%) '/gate_path/u14/OPC_reg_2_/' (31460) m8051_5ch_chain26 3  
// 13 14(34.15%) '/gate_path/u4/ACLDAT_reg_3_/' (31486) m8051_5ch_chain32 5  
// 14 14(34.15%) '/gate_path/u5/TMPDAT_reg_3_/' (31495) m8051_5ch_chain27 4  
// 15 13(31.71%)  '/gate_path/u5/TMPDAT_reg_4_/' (31494) m8051_5ch_chain27 5  
// 16 12(29.27%)  '/gate_path/u14/OPC_reg_7_/' (31455) m8051_5ch_chain26 8  
// 17 12(29.27%)  '/gate_path/u1/Q5_reg/' (31158) m8051_5ch_chain23 7  
// 18 12(29.27%)  '/gate_path/u15/u10_DAT_reg_4_/' (31266) m8051_5ch_chain40 4 ATPG  
//                                     constraint on  
//                                     '/gate_path/u15/u10_DAT_reg_4_/Q' (1458)  
// 19 11(26.83%)  '/gate_path/u4/L_ACCDAT_reg_7_/' (31464) m8051_5ch_chain30 7  
// 20 11(26.83%)  '/gate_path/u15/u10_DAT_reg_5_/' (31267) m8051_5ch_chain40 3 ATPG  
//                                     constraint on  
//                                     '/gate_path/u15/U314/Z' (10147)  
// 21 10(24.39%)  '/gate_path/u4/L_ACCDAT_reg_6_/' (31465) m8051_5ch_chain30 6  
// 22 10(24.39%)  '/gate_path/u7/INDIRECT_ADDR_reg_7_/' (31347) m8051_5ch_chain1 6 Cell  
//                                     constraint  
// 23 10(24.39%)  '/gate_path/u5/TMPDAT_reg_7_/' (31491) m8051_5ch_chain27 8  
// 24 10(24.39%)  '/gate_path/u5/TMPDAT_reg_5_/' (31493) m8051_5ch_chain27 6
```

```

//25 9(21.95%) '/gate_path/u4/L_ACCDAT_reg_5_/' (31466) m8051_5ch_chain30 5
//26 9(21.95%) '/gate_path/u4/L_ACCDAT_reg_4_/' (31467) m8051_5ch_chain30 4
//27 9(21.95%) '/gate_path/u15/u10_DAT_reg_1_/' (31263) m8051_5ch_chain40 7 ATPG
//                                         constraint on
//                                         '/gate_path/u15/u10_DAT_reg_1_/O' (1455)
//28 9(21.95%) '/gate_path/u5/TMPDAT_reg_2_/' (31496) m8051_5ch_chain27 3
//29 8(19.51%) '/gate_path/u7/L_FA_reg_4_/' (31343) m8051_5ch_chain5 7
//30 8(19.51%) '/gate_path/u7/INDIRECT_ADDR_reg_6_/' (31348) m8051_5ch_chain1 5 Cell
//                                         constraint
//
//Top specified shift positions are:
//Shift position Specified times Average specified times per cube
//-----
//7          106           2
//5          96            2
//6          91            2
//4          87            2
//3          68            1
//
// Total analysis time = 0.00 sec.

```

In the above example, we focus on the cubes whose size is below or near encoding capacity. As noted earlier, EAB cubes are ordered in ascending order of their size in the failing block. The tool listed 30 top specified scan cells. For each scan cell, the tool reports its specified times among the selected EAB cubes, gate name and id, scan chain, shift position counted from scan input, block name and known constraint on the gate.

From the constraint column, we can see that a number of top specified bits have constraints on them. For example, the 4th scan cell has a cell constraint, the 2nd one is a condition bit of an NCP and so on. You may see the following constraint types in this column:

- **Cell Constraint** — A C0/C1 cell constraint is on this scan cell.
- **ATPG Constraint** — The scan cell's value is determined due to some added atpg constraints. Note, the atpg constraint may not be added on the cell directly. E.g. if an AND gate is constrained to 1, both of its inputs must be 1. If one of the inputs happens to be a scan cell, it will have this constraint.
- **Condition Bit of an NCP** — This scan cell needs to load a pre-determined value for the NCP.
- **Condition Bit of a Clock Under Control** — The bit is part of the clock control definition of an internal clock.
- **Equivalent Scan Cell** — The scan cell is equivalent to another scan cell that is constrained.
- **Design Constraint** — The scan cell does not have a “hard” constraint. However, when we try to satisfy all the ATPG constraints and prevent all the bus contentions in the design, this scan cell is part of the solution. If this type of scan cell is in the top list, the encoding problem may come from busses or ATPG constraints in the design.

Example 3

The following example shows a design that suffers from excessive EAB faults due to clock control condition bits.

Note that in this example, the “Gate” column entries are shortened with */gate_path*, which represents the following:

/core_0/dig_core_0/wrap_0/test_clock_ctrl_test_clk_bp_0

This is done for presentation purposes.

ANALYSIS> report_edt_abort_analysis

```

//  

// EAB test cube analysis results:  

// EDT configurations:  

// Number of channels      = 10  

// Number of scan chains    = 800  

// Number of scan cells     = 90766  

// Longest scan chain length = 183  

// Approximate encoding capacity = 1387  

//  

// EAB test cube size distribution:  

// Specified bits   Specified bits / Capacity (%)   EAB cubes  

// -----  

// 277 - 416        20.00% - 30.00%                81  

// > 1387           > 100%                         19  

// Total             -                                100  

//  

// Analyzed a total of 100 EAB cubes.  

// Top specified scan cells are:  

// ID  Specified times  Gate  

// ID Specified  Gate          Chain      Shift  Constraint  

// times          -          position  

// ---  

// 1  81(81.00%) '/gate_path/ShiftReg/FF_reg[3]/' (4622541)  chain772  0  Condition of  

//                                         Clock Control for  

//                                         '/hsp_test_clk_bp_int'(117)  

// 2  80(80.00%) '/gate_path/ShiftReg/FF_reg[2]/' (4595006)  chain769  1  Condition of  

//                                         Clock Control for  

//                                         '/test_clk_bp_int' (112)  

// 3  79(79.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4595005)  chain769  2  Condition of Clock  

//                                         Control for  

//                                         '/test_clk_bp_int' (112)  

// 4  77(77.00%) '/gate_path/shiftReg/FF_reg[0]/' (4597960)  chain775  3  Condition of  

// Clock  

//                                         Control for  

//                                         '/test_clk_bp_int' (110)  

// 5  75(75.00%) '/gate_path/ShiftReg/FF_reg[0]/' (4542701)  chain774  3  ATPG constraint on  

//                                         '/gate_path/ShiftReg/FF_reg[0]/Q' (273991)  

// 6  74(74.00%) '/gate_path/ShiftReg/FF_reg[2]/' (4598252)  chain770  1  Condition of Clock  

//                                         Control for  

//                                         '/test_clk_bp_int' (111)  

// 7  65(65.00%) '/gate_path/ShiftReg/FF_reg[3]/' (4599800)  chain767  0  Condition of Clock  

//                                         Control for  

//                                         '/test_clk_bp_int' (109)  

// 8  63(63.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4542702)  chain774  2  ATPG constraint on  

//                                         '/gate_path/ShiftReg/FF_reg[1]/Q' (273992)  

// 9  63(63.00%) '/gate_path/ShiftReg/FF_reg[0]/' (4623366)  chain765  3  Condition of Clock  

//                                         Control for  

//                                         '/p_test_clk_bp_int' (108)  

// 10 60(60.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4598251)  chain770  2  Condition of Clock  

//                                         Control for  

//                                         '/test_clk_bp_int' (111)  

//  

// Top specified shift positions are:  

// Shift position  Specified times  Average specified times per cube  

// -----  

// 3              592            5  

// 0              511            5  

// 1              491            4

```

```
//   2           425      4
//   4           22       0
//   5           17       0
//  15           16       0
//   7           16       0
//  16           15       0
//   6           15       0
//
//  Total analysis time = 2.24 sec.
```

Example 4

Another usage of the command is to inspect specific test cubes.

Note that in this example, the “Gate” column entries are shortened with */gate_path*, which represents the following:

/m8051_5channels/m8051_i

This is done for presentation purposes.

ATPG> report_edt_abort_analysis -block m8051_5ch -from 1 -to 3 -cubes

```
//EAB test cube #1:
//Gate                                Value Chain          Shift  Constraint
//                                         pos
//                                         (from sci)
-----
//'/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) 1    m8051_5ch_chain1  3    Cell
//                                         constraint
//'/gate_path/u14/OPC_reg_2_/' (31460)   0    m8051_5ch_chain26  3
//'/gate_path/u14/OPC_reg_4_/' (31458)   1    m8051_5ch_chain26  5
//'/gate_path/u4/ACLDAT_reg_6_/' (31483)  1    m8051_5ch_chain32  8
//'/gate_path/u5/TMPDAT_reg_6_/' (31492)  1    m8051_5ch_chain27  7
//'/gate_path/u7/L_FA_reg_4_/' (31343)   1    m8051_5ch_chain5   7
//'/gate_path/u4/L_ACCDAT_reg_4_/' (31467) 0    m8051_5ch_chain30  4
//'/gate_path/u4/L_ACCDAT_reg_7_/' (31464) 0    m8051_5ch_chain30  7
// Total bits specified = 8
//
// EAB test cube #2:
//Gate                                Value Chain          Shift  Constraint
//                                         pos
//                                         (from sci)
//-----
//'/gate_path/u15/u10_DAT_reg_4_/' (31266) 0    m8051_5ch_chain40  4    ATPG
//                                         constraint on
//                                         '/gate_path/u15/u10_DAT_reg_4_/_Q' (1458)
//'/gate_path/u15/u10_DAT_reg_6_/' (31268)  0    m8051_5ch_chain40  2    ATPG
//                                         constraint on
//                                         '/gate_path/u15/u10_DAT_reg_6_/_Q' (1460)
//'/gate_path/u15/u10_DAT_reg_5_/' (31267)  1    m8051_5ch_chain40  3    ATPG
//                                         constraint on
//                                         '/gate_path/u15/U314/Z' (10147)
//'/gate_path/u7/INDIRECT_ADDR_reg_2_/' (31352) 1    m8051_5ch_chain1   1    Cell
//                                         constraint
//'/gate_path/u4/L_ACCDAT_reg_3_/' (31468)  1    m8051_5ch_chain30  3
//'/gate_path/u14/OPC_reg_0_/' (31462)   0    m8051_5ch_chain26  1
//'/gate_path/u14/OPC_reg_4_/' (31458)   1    m8051_5ch_chain26  5
//'/gate_path/u5/TMPDAT_reg_2_/' (31496)  1    m8051_5ch_chain27  3
//'/gate_path/u4/ACLDAT_reg_2_/' (31487) 0    m8051_5ch_chain32  4
// Total bits specified = 9
//
//EAB test cube #3:
//Gate                                Value Chain          Shift  Constraint
//                                         pos
//                                         (from sci)
//-----
//'/gate_path/u15/u10_DAT_reg_3_/' (31265) 0    m8051_5ch_chain40  5    ATPG
//                                         constraint on
//                                         '/gate_path/u15/u10_DAT_reg_3_/_Q' (1457)
//'/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) 1    m8051_5ch_chain1   3    Cell
//                                         constraint
//'/gate_path/u14/OPC_reg_2_/' (31460)   0    m8051_5ch_chain26  3
//'/gate_path/u14/OPC_reg_4_/' (31458)   1    m8051_5ch_chain26  5
//'/gate_path/u14/OPC_reg_5_/' (31457)   0    m8051_5ch_chain26  6
//'/gate_path/u5/TMPDAT_reg_6_/' (31492)  1    m8051_5ch_chain27  7
//'/gate_path/u5/TMPDAT_reg_5_/' (31493)  0    m8051_5ch_chain27  6
//'/gate_path/u4/L_ACCDAT_reg_5_/' (31466) 1    m8051_5ch_chain30  5
```

```
///'/gate_path/u1/Q4_reg/' (31164)          0      m8051_5ch_chain23   6
///'/gate_path/u4/L_ACCDAT_reg_6_/' (31465)  0      m8051_5ch_chain30   6
///'/gate_path/u4/L_ACCDAT_reg_4_/' (31467)  1      m8051_5ch_chain30   4
// Total bits specified = 11
//
// Total analysis time = 0.00 sec.
```

The test cubes are represented in a similar way as the cells in the top specified cell list. A little different is that the bit in an EAB cube has a value field. We can see that the number of EAB cubes is pretty small. For example, the first reported EAB cube contains only 8 bits. In certain situations such as linear dependency, it may not be possible to encode these 8 bits even though the encoding capacity of the block is over 60. If detection of a fault requires specification of these 8 bits, the fault may be classified as EAB.

Example 5

When multiple EDT blocks share the same EDT channel input pins, analysis of EAB cubes must consider such blocks together.

Note that in this example, the “Gate” column entries are shortened with */gate*, which represents the following:

/EngineX4_i

This is done for presentation purposes.

The generated report displays such blocks as a composite EDT block named COMPOSITE_BLOCK_N. The following example shows EAB analysis for 6 EDT blocks that share EDT input channels and are reported as COMPOSITE_BLOCK_0 and another 2 blocks reported as COMPOSITE_BLOCK_1:

ANALYSIS> report_edt_abort_analysis

```

//
// EAB test cube analysis results for composite EDT block 'COMPOSITE_BLOCK_0':
// This composite EDT block contains the following EDT blocks:
// blk1
// blk2
// blk7
// blk8
// blk3
// blk4
// EDT configurations:
// Number of channels      = 13
// Number of scan chains    = 600
// Number of scan cells     = 18899
// Longest scan chain length = 32
// Approximate encoding capacity = 310
//
// EAB test cube size distribution:
// Specified bits   Specified bits / Capacity (%)   EAB cubes
// -----
// 31 - 62          10.00% - 20.00%                 3
// 62 - 93          20.00% - 30.00%                35
// 93 - 124         30.00% - 40.00%                 5
// 124 - 155        40.00% - 50.00%                 3
// Total           100.00%                          46
//
// Analyzed a total of 46 EAB cubes.
// Top specified scan cells are:
// ID Specified Gate                                         Chain Shift
// times                                                 position
// --- -----
// 1 4 (8.70%) '/gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[29]5/' (574905) 387 30
// 2 4 (8.70%) '/gate/Engine_Instance_4/SE_PEO/V2Y_reg[8]/' (584285) 81 26
// 3 3 (6.52%) '/gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[0]5/' (574913) 387 22
// 4 3 (6.52%) '/gate/Engine_Instance_4/SE_PEO/V2Y_reg[22]/' (582973) 62 28
// 5 3 (6.52%) '/gate/Engine_Instance_2/SE_EX0/COO_REG_OUT_SEL_reg/' (568317) 628 17
// 6 3 (6.52%) '/gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[2]5/' (574906) 387 29
// 7 3 (6.52%) '/gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[22]5/' (574907) 387 28
// 8 3 (6.52%) '/gate/Engine_Instance_4/SE_PEO/V1Y_reg[1]/' (583376) 68 26
// 9 3 (6.52%) '/gate/Engine_Instance_4/SE_PEO/V1Y_reg[0]/' (584345) 82 26
// 10 3 (6.52%) '/gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[4]/Q' (579371) 366 3
//
// Top specified shift positions are:
// Shift position  Specified times  Average specified times per cube
// --- -----
// 26             73                  1
// 28             54                  1
// 27             48                  1
// 21             38                  0
// 30             36                  0
// 18             35                  0
// 17             34                  0
// 25             33                  0
// 23             32                  0
// 20             31                  0
//
// EAB test cube analysis results for composite EDT block 'COMPOSITE_BLOCK_1':
// This composite EDT block contains the following EDT blocks:
// blk5

```

Command Dictionary (R) report_edt_abort_analysis

```
// blk6
// EDT configurations:
// Number of channels      = 3
// Number of scan chains    = 200
// Number of scan cells     = 6253
// Longest scan chain length = 32
// Approximate encoding capacity = 89
//
// EAB test cube size distribution:
// Specified bits  Specified bits / Capacity (%)  EAB cubes
// -----
// 35 - 44          40.00% - 50.00%                1
// 71 - 80          80.00% - 90.00%                1
// 80 - 89          90.00% - 100.00%               1
// > 89             > 100%                      6
// Total                         9
//
//Analyzed a total of 9 EAB cubes.
//Top specified scan cells are:
//ID Specified  Gate                                         Chain Shift
// times                                         position
//-----
//1 3(33.33%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[70]1/Q' (579854) 437 10
//2 3(33.33%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[97]1/Q' (579826) 436 7
//3 2(22.22%)   '/gate/Engine_Instance_2/SE_PEO/ZSS_P_reg[10]3/Q' (571404) 505 26
//4 2(22.22%)   '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[33]1/Q' (579889) 438 7
//5 2(22.22%)   '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[28]6/Q' (579885) 438 11
//6 2(22.22%)   '/gate/Engine_Instance_2/SE_PEO/N1523_reg/Q' (570880) 488 15
//7 2(22.22%)   '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[57]1/Q' (579839) 437 25
//8 2(22.22%)   '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[25]6/Q' (579882) 438 14
//9 2(22.22%)   '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[31]6/Q' (579881) 438 15
//10 2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[6]6/Q' (579880) 438 16
//
// Top specified shift positions are:
// Shift position  Specified times  Average specified times per cube
// -----
// 26            17                  1
// 11            12                  1
// 28            11                  1
// 7              9                  1
// 16            9                  1
// 25            8                  0
// 10            8                  0
// 21            8                  0
// 24            7                  0
// 15            7                  0
//
// EAB test cube analysis results for EDT block 'blk9':
// EDT configurations:
// Number of channels      = 1
// Number of scan chains    = 100
// Number of scan cells     = 3084
// Longest scan chain length = 32
// Approximate encoding capacity = 42
//
// EAB test cube size distribution:
// Specified bits  Specified bits / Capacity (%)  EAB cubes
// -----
```

```

// 33 - 37      80.00% - 90.00%          3
// 37 - 42      90.00% - 100.00%         5
// > 42          > 100%                  97
// Total          105

//
// Analyzed a total of 105 EAB cubes.
// Top specified scan cells are:
// ID Specified times Gate //
ID Specified   Gate                                Chain Shift
// times                                         position
//----- -----
//1 14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/Z_reg[14]1/Q' (565444) 880 18
//2 14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[4]/Q' (565169) 814 18
//3 14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[3]/Q' (565104) 812 22
//4 12(11.43%) '/gate/Engine_Instance_1/SE_EX0/PX_CO01_reg[16]8/' (561265) 854 10
//5 12(11.43%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[25]/Q' (565260) 817 18
//6 11(10.48%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[13]/Q' (565077) 811 19
//7 10(9.52%)  '/gate/Engine_Instance_1/SE_EX0/PX_CO01_reg[28]8/' (561266) 854 9
//8 10(9.52%)  '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[21]/Q' (565223) 816 25
//9 10(9.52%)  '/gate/Engine_Instance_1/SE_EX0/COO_REG_OUT_SEL_reg/' (561258) 854 17
//10 9(8.57%)  '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[2]/Q' (565196) 815 21
//
// Top specified shift positions are:
// Shift position Specified times Average specified times per cube
//----- -----
// 19           94           0
// 18           84           0
// 15           81           0
// 22           75           0
// 21           72           0
// 25           68           0
// 10           68           0
// 13           66           0
// 17           66           0
// 16           65           0
//
// Total analysis time = 0.38 sec.

```

Example 6

The following example reports EAB statistics for COMPOSITE_BLOCK_0 from the previous example.

Note that in this example, the “Gate” column entries are shortened with */gate*, which represents the following:

/EngineX4_i

This is done for presentation purposes.

ANALYSIS> report_edt_abort_analysis -block COMPOSITE_BLOCK_0 -cubes -from 1 -to 2

Command Dictionary (R) **report_edt_abort_analysis**

```
//EAB test cube #1:  
//Gate  
//  
//  
//-----  
///'gate/Engine_Instance_3/SE_EX0/REG_OUT_SEL_reg25/Q' (574882) 0 386 22  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[18]5/' (574910) 1 387 25  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[27]5/' (574849) 1 385 23  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[26]5/' (574915) 0 387 20  
///'gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[3]/Q' (579096) 0 314 26  
///'gate/Engine_Instance_3/SE_FMUL00/C_FMUL_reg[3]/Q' (579320) 0 365 22  
///'gate/Engine_Instance_3/SE_FMUL00/C_FMUL_reg[0]/Q' (579252) 0 363 27  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[24]2/' (573291) 0 320 27  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[8]5/' (574852) 1 385 20  
//Total bits specified = 9  
//  
//EAB test cube #2:  
//Gate  
//  
//  
//-----  
///'gate/Engine_Instance_3/SE_PEO/DX23_reg[9]/' (576871) 0 300 30  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[20]5/' (574908) 1 387 27  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[0]5/' (574913) 1 387 22  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[14]5/' (574850) 1 385 22  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[29]5/' (574905) 0 387 30  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[18]5/' (574910) 1 387 25  
///'gate/Engine_Instance_3/SE_EX0/PX_CO00_reg[22]5/' (574907) 0 387 28  
///'gate/Engine_Instance_3/SE_EX0/REG_OUT_SEL_reg28/Q' (575097) 1 393 28  
///'gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[0]/Q' (579345) 1 366 29  
//Total bits specified = 9  
//  
// Total analysis time = 0.00 sec.
```

Related Topics

[set_edt_abort_analysis_options](#)

report_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.

Usage

`report_edt_blocks`

Description

Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.

When the display shows “current block” by a name, it indicates relevant tool commands apply only to the EDT block associated with that name.

You can transfer this restriction (also referred to as the EDT context) to another EDT block with the [set_current_edt_block](#) or [add_edt_blocks](#) commands.

Arguments

None

Examples

The following example lists currently defined EDT block names:

```
report_edt_blocks
// my_core1_block
// my_core2_block
// my_core3_block
// my_core4_block    (current block)
```

Related Topics

[add_edt_blocks](#)

[delete_edt_blocks](#)

[set_current_edt_block](#)

report_edt_configurations

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays configuration information for the current compression logic.

Usage

```
report_edt_configurations [-ALL_blocks] [<> | >>] file_pathname]
```

Description

Displays configuration information for the current compression logic.

Among the items reported are the version, compactor type, power controller status, low-power pipeline stages, number of scan channels and related information, multiple compression configurations, and whether the tool is set up to insert bypass logic and lockup cells. This command only displays a partial report from Setup mode. To produce a complete report, you must exit Setup mode to run DRC and flatten the model before issuing this command.

Note

 The report_edt_configurations command does not list the names of EDT pins. To display the names of EDT pins, use the [report_edt_pins](#) command.

The -all_blocks switch is useful where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the EDT configuration information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -all_blocks switch, the command displays configuration information for every EDT block. You set the current EDT block using either the [add_edt_blocks](#) or [set_current_edt_block](#) command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the [Modular Compressed ATPG](#) chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- `-ALL_blocks`

An optional switch that displays EDT configuration information for all EDT blocks. Without this switch, only information for the current EDT block displays. If a current EDT block is not defined, configuration information for all EDT blocks displays.

- `>file_pathname`

An optional redirection operator and pathname pair for creating or replacing the contents of `file_pathname`.

- `>>file_pathname`

An optional redirection operator and pathname pair for appending to the contents of `file_pathname`.

Examples

Example 1

The following example reports configuration details for a non-modular design with eight scan chains and one scan channel; first while in Setup mode, then again after DRC:

```
report_edt_configurations

// IP version: 1
// External scan channels: 1
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive

set_system_mode analysis

...
# -----
# Begin EDT rules checking.
# -----
# Running EDT logic creation Phase.
# 5 pin(s) will be added to the EDT wrapper. (K11)
# EDT rules checking completed, CPU time=0.01 sec.
# All scan input pins were forced to TIE-X.
# All scan output pins were masked.
# -----
...

report_edt_configurations

// IP version: 1
// Shift cycles: 381, 373 (internal scan length)
// + 8 (additional cycles)
// External scan channels: 1
// Internal scan chains: 8
// Masking bits: 2
// Longest chain range: 2 - 373
// Decompressor size: 32
// Shift power controller: Enabled and inactive(shift power control
// is off)
// Min switching threshold: 25%
// Compactor type: Xpress
// Scan cells: 18650
// Compression per pattern: 16.01x (ATPG bypass = 3 x 6341, EDT = 3 x
// 396)
// Clocking: edge-sensitive
// Compactor pipelining: Off
```

Example 2

The following example defines two EDT blocks in the patterns -scan context, then reports configuration details first for the current block and then for all blocks:

```
# Define the block name "my_block1" for an EDT block
# and automatically set it as the current EDT block.
```

```
add_edt_blocks my_block1
# Define the location (instance name) of the top-level EDT logic for this
#   block; also the decompressor and compactor.

set_edt_instances -edt_logic_top core1_edt_i
set_edt_instances -decompressor core1_edt_decompressor_i
set_edt_instances -compactor core1_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 3

# Specify the top-level control signals and channel inputs and outputs for
#   this block (required).

set_edt_pins...

# Repeat the preceding procedure for another block.

add_edt_block my_block2
set_edt_instances -edt_logic_top core2_edt_i
set_edt_instances -decompressor core2_edt_decompressor_i
set_edt_instances -compactor core2_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 1
set_edt_pins...

report_edt_configurations

// EDT block "my_block2"
// -----
// IP version:                      3
// External scan channels:          1
// Decompressor size:               10
// Compactor type:                 Xpress
// Clocking:                        edge-sensitive
//
// Note: Only current EDT block "my_block2" is reported. Use "report edt
// configuration -all_blocks" to report the configurations of all blocks.
```

report_edt_configurations -all_blocks

```
// EDT block "my_block1"
// -----
// IP version: 3
// External scan channels: 3
// Decompressor size: 12
// Compactor type: Xpress
// Clocking: edge-sensitive
//
// EDT block "my_block2"
// -----
// IP version: 3
// External scan channels: 1
// Decompressor size: 10
// Compactor type: Xpress
// Clocking: edge-sensitive
```

Example 3

The following example reports all EDT blocks. Channel sharing information is shown in red font. Note that “// Blocks sharing input channels:” means the blocks having at least one input channel that is shared with this channel.

report_edt_configurations -all

Command Dictionary (R) **report_edt_configurations**

```
// EDT block "B1"
// -----
// IP version: 4
// Shift cycles: 53, 19 (internal scan length) + 11
// (additional cycles) + 23 (pad cycles due to blocks with longer chains)
// External scan channels: 2 (1 control input channel)
// Internal scan chains: 32
// Masking bits: 25
// Longest chain range: 2 - 19
// Decompressor size: 16
// Compactor type: Xpress
// Blocks sharing input channels: B2
// Scan cells: 573
// Clocking: edge-sensitive
// Compactor pipelining: 0 stages
//
// EDT block "B2"
// -----
// IP version: 4
// Shift cycles: 53, 2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
// External input channels: 3 (1 control input channel)
// External output channels: 2
// Internal scan chains: 4
// Masking bits: 5
// Longest chain range: 2 - 2
// Decompressor size: 8
// Compactor type: Xpress
// Blocks sharing input channels: B1, B3
// Scan cells: 8
// Clocking: edge-sensitive
// Compactor pipelining: 0 stages
//
// EDT block "B3"
// -----
// IP version: 4
// Shift cycles: 53, 2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
// External scan channels: 2 (1 control input channel)
// Internal scan chains: 4
// Masking bits: 5
// Longest chain range: 2 - 2
// Decompressor size: 8
// Compactor type: Xpress
// Blocks sharing input channels: B2
// Scan cells: 8
// Clocking: edge-sensitive
// Compactor pipelining: 0 stages
```

Example 4

In this example, the tool reports while in non-setup mode if there is an EDT low-power controller:

report_edt_configurations

```
//  
// IP version: 6  
// Shift cycles: 148, 143 (internal scan length) + 5 (additional cycles)  
// External scan channels: 23 (4 control input channels)  
// Internal scan chains: 1827  
// Masking bits: 254  
// Decompressor size: 92 (4 segments)  
// Scan cells: 258561  
// Compression per pattern: 76.33x (ATPG bypass = 23 x 11297, EDT = 23 x 148)  
// Bypass logic: On  
// Lockup cells: On  
// Clocking: edge-sensitive  
// Compactor pipelining: Off  
// Low power shift controller: Enabled and active  
//   Min switching threshold: 25%  
//   Low power bits: 30
```

Related Topics

[add_edt_configurations](#)
[report_scan_volume](#)
[report_edt_pins](#)
[delete_edt_configurations](#)
[set_edt_options](#)
[set_edt_pins](#)

report_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports information about the EDT logic contained in a design.

Usage

To report decompressors

```
report_edt_finder -Decompressors [-Type {ACtive | Inactive | ALL}]  
[-Id decompressor_id#] [-Filter instance_name_wildcard]  
[-Verbose [instances]] [{>} | >>] file_pathname
```

To report scan chain segments

```
report_edt_finder -Chain_segments  
[-Id decompressor_id#] [-Type {ALL | ACcessible | COMpressed | Uncompressed  
| CONtrol | Inaccessible}] [-Min_length integer] [-Max_length integer] [-Bins integer]  
[-Endpoints {instance_name_wildcard | gate_id#}]  
[-Details {SUMmary | Endpoints | Full}] [{>} | >>] file_pathname
```

To report masking logic

```
report_edt_finder -Masking_logic [-Id decompressor_id#] [-Filter instance_name_wildcard]  
[-Verbose [instances]] [{>} | >>] file_pathname
```

To report low-power shift controller logic

```
report_edt_finder -Low_power_shift_controller [-Id decompressor_id#]  
[-Filter instance_name_wildcard] [-Verbose [instances]] [{>} | >>] file_pathname
```

Description

Reports information about the EDT logic contained in a design. Depending on the specified switches, the following columns can be reported:

- Length — Specifies the length of the identified scan chains. For example, (16-101) means the smallest scan chain has 16 cells and the longest scan chain has 101 cells.
- Count — Specifies the number of scan chains. For example, the first line of this report shows there are 40 chains that are compressed with length 16-101.

```
// Type          Length        Count  
// -----  
// compressed    16-101      40  
// compressed    102-188     50  
// inaccessible  1-94       7520  
// inaccessible  95-189      1160
```

- Type — Specifies the type of chain segment as defined by the Type argument description.

- Begin/End — Specifies the first scan cell or last scan cell of the scan chain segment. Defined by the -Details Endpoints switch.

Arguments

- **-Decompressors**

A required switch that reports on decompressors found in the design.

- [-Type {ACtive| Inactive | ALl}]

An optional switch and literal pair that specifies which decompressors to report on. Options include:

ACtive — Reports on all active decompressors identified. This is the invocation default.

Inactive — Reports on all inactive decompressors identified (decompressors that do not reach channel outputs).

ALl — Reports all decompressors identified.

- **-Chain_segments**

A required switch that reports on scan chain segments found in the design.

- [-Type {ALl | ACcessible [Compressed | Uncompressed | CONtrol] | Inaccessible}]}

An optional switch and literal pair that specifies the type of chain segments to report. Literal options include:

ALl — Reports all chain segments found. This is the invocation default.

ACcessible — Reports all scan chain segments accessible via identified EDT logic or primary input/output pins. You can also enter an additional literal option to further sort on the accessible scan chains returned. Literal sort options include:

COMpressed — Reports compressed chains, such as chains accessible via identified EDT logic.

Uncompressed — Reports uncompressed chains such as chains accessible via primary input/output pins.

CONtrol — Reports chain segments of the EDT control logic such as the mask shift registers of Xpress compactors.

Inaccessible — Reports chain segments that are not accessible via identified EDT logic or primary input/output pins.

- **-MIn_length *integer***

An optional switch and integer pair that specifies the minimum length of chain segments to display. The default is 1.

- **-MAx_length *integer***

An optional switch and integer pair that specifies the maximum length of chain segments to display. The default is unlimited.

- **-Endpoints {*instance_name_wildcard* | *gate_id#*}**
An optional switch and string or integer that specifies the endpoints of the chain segments to display. The string may include asterisk (*) and/or question mark (?) wildcard characters.
- **-Details {SUMmary | Endpoints | Full}**
An optional switch and literal pair that specifies the details to display. Options include:
 - SUMmary — Reports only a summary such as the number of chains that belong to a certain category. This is the invocation default.
 - Endpoints — Reports the endpoints of the chain segments.
 - Full — Reports all available information about the chain segments, including the instance name of each chain element.
- **-Bins *integer***
An optional switch and integer pair that specifies the maximum number of bins that scan chain data is sorted by in the summary report.
Each type (accessible/compressed/uncompressed/control/inaccessible) of scan chain reported is sorted by scan chain lengths determined by the number of bins specified with this switch. By default, 10 bins are used.
- **-Masking_logic**
A required switch that reports details about specified Xpress mask registers identified by EDT Finder.
- **-Low_power_shift_controller**
A required switch that reports details about low-power shift and hold registers identified by EDT Finder.
- **-Id *decompressor_id#***
An optional switch and integer pair that specifies the identification number of the decompressor to report. The value of the *decompressor_id#* is the unique identification number automatically assigned to every decompressor by EDT Finder.
- **-Filter*instance_name_wildcard***
An optional switch and string pair that specifies the instance name of the decompressors or masking logic to report. You can use regular expressions that include asterisk (*) and/or question mark (?) wildcard characters.
- **-Verbose [instances]**
An optional switch and literal that transcribes informational messages about the EDT logic found in a design. Depending on the EDT logic specified, this switch reports informational messages about decompressors, low power registers, and masking logic found in the design. If the *instances* literal is specified, only the names of instances are reported. By default, these messages are disabled.

- **>file.pathname**

An optional redirection operator and pathname pair that creates or replaces the contents of *file.pathname* with the output.

- **>>file.pathname**

An optional redirection operator and pathname pair that appends the output to the contents of *file.pathname*.

Examples

Example 1

The following example displays a summary of decompressors found in a design:

report_edt_finder -decompressors

```
// Decomp. Id# #Bits #Inputs #Chains Block Type
// -----
// 1 16 4 28 m1_28x16 active
// 2 16 4 4 m2_4x32 active
// 3 16 4 50 m3_50x187 active
// 4 10 1 8 m4_8x16 active
```

Example 2

The following example displays a summary of all the masking logic found:

report_edt_finder -masking_logic

```
// Decomp. Id# Length Type Block
// -----
// 1 9 xpress piccpu_40
// 2 31 xpress my_core_5ch
// 3 31 xpress my_core_5ch_2
```

Example 3

This example displays a summary of all found chain segments grouped into 2 bins per type:

report_edt_finder -chain_segments -bins 2

```
// Type Length Count
// -----
// compressed 16-101 40
// compressed 102-188 50
// inaccessible 1-94 7520
// inaccessible 95-189 1160
```

Example 4

The following example displays the scan inputs and outputs of the accessible chain segments with a minimum length of 50:

report_edt_finder -chain_segments -type accessible compressed -min_length 50

```
// EDT Control          Length  Type           Name
// -----
// edt controllable    50     compressed   picccpu_90_chain2
// edt controllable    90     compressed   picccpu_90_chain3
// edt controllable    50     compressed   picccpu_90_2_chain2
// edt controllable    90     compressed   picccpu_90_2_chain3
```

Example 5

The following example displays a summary of the low-power shift controller logic in blk1:

```
report_edt_finder -low_power_shift_controller -id 2 -verbose instances
```

```
// Decomp. Id # Registers      Block
// -----
// 2             5               blk1
// Shift reg.  /top/blk1_i/.../low_power_shift_reg_0_reg[4]/UD1 (16171)
// Hold reg.   /top/blk1_i/.../control_hold_reg_0_reg[4]/DFF1 (16166)
// Shift reg.  /top/blk1_i/.../low_power_shift_reg_0_reg[3]/UD1 (16172)
// Hold reg.   /top/blk1_i/.../low_power_hold_reg_0_reg[3]/DFF1 (16167)
// Shift reg.  /top/blk1_i/.../low_power_shift_reg_0_reg[2]/UD1 (16173)
// Hold reg.   /top/blk1_i/.../low_power_hold_reg_0_reg[2]/DFF1 (16168)
// Shift reg.  /top/blk1_i/.../low_power_shift_reg_0_reg[1]/UD1 (16174)
// Hold reg.   /top/blk1_i/.../low_power_hold_reg_0_reg[1]/DFF1 (16169)
// Shift reg.  /top/blk1_i/.../low_power_shift_reg_0_reg[0]/UD1 (16175)
// Hold reg.   /top/blk1_i/.../low_power_hold_reg_0_reg[0]/DFF1 (16170)
```

Related Topics

[set_edt_finder](#)

report_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.

Usage

```
report_edt_instances [-All_blocks] [-Errors_only] [{> | >>} file.pathname]
```

Description

Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.

Note

 For modular Tessent TestKompress, the report_edt_instances command operates only on the current EDT block (the EDT block that is the current target of EDT-specific commands). This is the EDT block specified by the last add_edt_blocks or last [set_current_edt_block](#) command, whichever you issued most recently.

The report_edt_instances command lists the names of the instances where the tool expects to find the top-level EDT logic, decompressor, and compactor. It also reports the pathnames of these instances when known. If the display lists any instance paths as “NOT FOUND,” the tool will not be able to provide the maximum amount of detail in the debugging information for K19-K22 DRC violations. You can use the [set_edt_instances](#) command to update the instance information.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the [add_edt_blocks](#) or [set_current_edt_block](#) command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

 This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display instance information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-Errors_only**

An optional switch that specifies to display instance information only if the tool is unable to positively identify at least one of the instance names. This could be because the tool either cannot find the instance name/path, or the instance name is not unique.

- **>file.pathname**

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file.pathname*.

- **>>file.pathname**

An optional redirection operator and pathname pair for appending to the contents of *file.pathname*.

Examples

The following example displays the current EDT instance data for a design, updates the information, then displays the updates:

```
report_edt_instances

// edt_logic_top
//   Setting: edt_i
//   Path    : NOT FOUND
//
// decompressor
//   Setting: edt_decompressor_i
//   Path    : NOT FOUND
//
// compactor
//   Setting: edt_compactor_i
//   Path    : NOT FOUND

set_edt_instances -edt_logic_top my_core_edt_i \
                  -decompressor my_core_edt_decompressor_i \
                  -compactor my_core_edt_compactor_i

// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
// Found compactor at path:
//   "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
//   "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

```
report_edt_instances
```

```
// edt_logic_top
//   Setting: my_core_edt_i
//   Path    : /my_core_edt_top/my_core_edt_i
//
// decompressor
//   Setting: my_core_edt_decompressor_i
//   Path    : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//   Setting: my_core_edt_compactor_i
//   Path    : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Related Topics

[add_edt_blocks](#)
[set_edt_instances](#)
[set_current_edt_block](#)

report_edt_lockup_cells

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Displays a tabular report of the lockup cells in the EDT logic.

Usage

```
report_edt_lockup_cells [-All_blocks] [{> | >>} file.pathname]
```

Description

Displays a tabular report of the lockup cells in the EDT logic.

The report_edt_lockup_cells command reports details about the lockup cells in the EDT logic.

The command only reports lockup cells that will be included in the generated IP. Specifically, when lockup cell insertion is turned off (which is not recommended), the report does not include any lockup cells.

Note

 During pattern generation, the command reports the expected lockup cells and not the actual lockup cells in the design.

The reported information is arranged in tables that cover the following:

Note

 If you invoke this command in Setup mode because EDT DRC failed, some data will not be available, and hence, will not be reported.

- Scan clocking — Details about the first and last scan cell in each scan chain. For each flip-flop (FF), the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO). The names of the first and last cell's clocks and their capture timing are also provided. The capture time is listed in the “Cap” column for the first cell, and in the “Chg” column for the last cell.
- Location — Each lockup cell is listed under one of the following categories depending on its location in the IP:
 - Decompressor—Lockup cells between the decompressor and the scan chain inputs
 - Compactor — Lockup cells between the scan chain outputs and the compactor
 - Bypass — Lockup cells in the bypass circuitry

- Type of lockup cell — Flip-flop (FF) or latch (LA). For each flip-flop, the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO).
- Clock — Name of the clock for each lockup cell
- Relevant Clocks — Summary of the off state, type, and timing of all clocks listed elsewhere in the report. The timing is based on the shift procedure in the test procedure file.

For more information, see “[Understanding Lockup Cells](#)” in the Tesson TestKompress User’s Manual.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tesson TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the lockup cell information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the [add_edt_blocks](#) or [set_current_edt_block](#) command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tesson TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

 This switch is only for the top-level Pattern Generation Phase of modular Tesson TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tesson TestKompress flow, you do not need this switch.

An optional switch that specifies to display the lockup cell information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- >*file.pathname*

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file.pathname*.

- >>*file.pathname*

An optional redirection operator and pathname pair for appending to the contents of *file.pathname*.

Examples

Example 1

The following example reports lockup details for a design with eight scan chains and one scan channel. The EDT logic is FF-based and includes bypass circuitry:

```
set_system_mode analysis
report_edt_lockup_cells

// SCAN Clocking
//-----+-----+
//           | First Cell      | Last Cell
// Chain (#)| Gate (id) Type Cap Clock|Gate (id) Type Chg Clock
//-----+-----+
// chain1(1)|/i7/ (92) FF-TE 20 /clk |/rg0/ (33) FF-TE 20 /clk
...
// chain8(8)|/i5/ (48) FF-TE 20 /clk |/ix8/ (89) FF-TE 20 /clk
//
// DECOMPRESSOR Lockup Cells
//-----+-----+
//           | First cell      | Second cell
// Chain (#)| Type   Clock      | Type   Clock
//-----+-----+
// chain1(1)| FF-TE /edt_clock | -----
// chain2(2)| FF-TE /edt_clock | FF-LE /edt_clock
...
// chain8(8)| FF-TE /edt_clock | -----
//
// COMPACTOR Lockup Cells
//-----+-----+
//           | Lockup cell
// Chain (#)| Type   Clock
//-----+-----+
// chain1(1)| FF-TE /edt_clock
...
// chain8(8)| FF-TE /edt_clock
//
// BYPASS Lockup Cells
//-----+-----+
//           | Lockup cell
// Chain (#) -> Chain (#) | Type   Clock
//-----Channel 1-----+-----+
// chain1 (1) -> chain2 (2) | FF-TE /clk
// chain2 (2) -> chain3 (3) | -----
// chain3 (3) -> chain4 (4) | -----
// chain4 (4) -> chain5 (5) | -----
// chain5 (5) -> chain6 (6) | -----
// chain6 (6) -> chain7 (7) | -----
// chain7 (7) -> chain8 (8) | -----
//
// RELEVANT Clocks
//-----+-----+-----+
//           | Off    |           | Time
// Name      | State  | Type     | LE TE
//-----+-----+-----+
// /edt_clock | 0     | EDT      | 20 30
// /clk       | 0     | Shift Master | 20 30
```

The following is the bypass portion of a report for a design with eight scan chains and two scan channels. Notice that in addition to listing the bypass lockups, the bypass tables show the chain connections to the channels.

```
// BYPASS Lockup Cells
//-----+-----+-----+
//          | Lockup cell
// Chain   (#) -> Chain   (#) | Type   Clock
//-----Channel 1-----+-----+
// chain1   (1) -> chain2   (2) | ----- -----
// chain2   (2) -> chain3   (3) | ----- -----
// chain3   (3) -> chain4   (4) | ----- -----
// chain4   (4) -> chain5   (5) | ----- -----
// chain5   (5) -> chain6   (6) | ----- -----
//-----Channel 2-----+-----+
// chain7   (7) -> chain8   (8) | ----- -----
// chain8   (8) -> chain9   (9) | ----- -----
// chain9   (9) -> chain11  (10) | FF-TE /NX1
// chain11  (10) -> chain10  (11) | FF-TE /NX2
```

Example 2

The following example shows a design with eight scan chains; four of the scan chains are clocked by LE clk1 and the other four scan chains are clocked by TE clk2. The report_edt_lockup_cells output that follows shows the inserted lockup cells at the scan chain inputs and outputs.

```
set_edt_options -lockup on -retime_chain_boundaries on
set_system_mode analysis
report_edt_lockup_cells
```

```
//  
// SCAN Clocking  
// -----+-----+-----+-----+-----+-----+  
// | First cell | Last cell  
// Chain (#) | Gate (id) Type Cap Clock | Gate (id) Type Chg Clock  
// -----+-----+-----+-----+-----+-----+  
// chain1 (1) | /chain1_cell119/Q (234) FF-LE 40 /clk1 | /chain1_cell10/Q (215) FF-LE 40 /clk1  
// chain2 (2) | /chain2_cell119/Q (254) FF-LE 40 /clk1 | /chain2_cell10/Q (235) FF-LE 40 /clk1  
// chain3 (3) | /chain3_cell119/Q (274) FF-LE 40 /clk1 | /chain3_cell10/Q (255) FF-LE 40 /clk1  
// chain4 (4) | /chain4_cell119/Q (294) FF-LE 40 /clk1 | /chain4_cell10/Q (275) FF-LE 40 /clk1  
// chain5 (5) | /chain5_cell119/Q (314) FF-TE 50 /clk2 | /chain5_cell10/Q (295) FF-TE 50 /clk2  
// chain6 (6) | /chain6_cell119/Q (334) FF-TE 50 /clk2 | /chain6_cell10/Q (315) FF-TE 50 /clk2  
// chain7 (7) | /chain7_cell119/Q (354) FF-TE 50 /clk2 | /chain7_cell10/Q (335) FF-TE 50 /clk2  
// chain8 (8) | /chain8_cell119/Q (374) FF-TE 50 /clk2 | /chain8_cell10/Q (355) FF-TE 50 /clk2  
//  
// -----+-----+-----+-----+-----+-----+  
// | Decompressor | Chain Input | Chain Output  
// | Lockup | Lockup | Lockup  
// Chain (#) | Type Clock | Type Clock | Type Clock  
// -----+-----+-----+-----+-----+-----+  
// chain1 (1) FF-TE /edt_clock ----- FF-TE /clk1  
// chain2 (2) FF-TE /edt_clock ----- FF-TE /clk1  
// chain3 (3) FF-TE /edt_clock ----- FF-TE /clk1  
// chain4 (4) FF-TE /edt_clock ----- FF-TE /clk1  
// chain5 (5) FF-TE /edt_clock FF-LE /clk2 -----  
// chain6 (6) FF-TE /edt_clock FF-LE /clk2 -----  
// chain7 (7) FF-TE /edt_clock FF-LE /clk2 -----  
// chain8 (8) FF-TE /edt_clock FF-LE /clk2 -----  
//  
// RELEVANT Clocks  
// -----+-----+-----+-----+  
// | Off | | Time  
// Name | state | Type | LE TE  
// -----+-----+-----+-----+  
// /edt_clock | 0 | EDT | 40 50  
// /clk1 | 0 | Shift MASTER | 40 50  
// /clk2 | 0 | System MASTER | 40 50
```

Related Topics

[report_edt_configurations](#)

[set_edt_options](#)

report_edt_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the currently defined names and inversion status of EDT channel and control pins.

Usage

```
report_edt_pins [-All_blocks] [-Group_by_pin_name]
[ {> | >>} file.pathname ]
```

Description

Displays the currently defined names and inversion status of EDT channel and control pins. The report_edt_pins command displays the names of all EDT channel and control pins and whether the pin signals are inverted between the chip input pin and the EDT logic.

The -All_blocks switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

 This switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display the EDT channel and control pin information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays configuration information for all EDT blocks.

- -Group_by_pin_name

An optional switch that specifies to display the pin name and all of the blocks that share that pin.

- >*file.pathname*

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file.pathname*.

- **>> file.pathname**

An optional redirection operator and pathname pair for appending to the contents of *file.pathname*.

Examples

Example 1

Assume the tool is in setup mode. The following example reports, for a design with two scan channels and no shared EDT pins, the names and inversion status of the EDT pins. The example then shares the EDT clock pin with a functional pin called “a1”. Following that, it renames the dedicated EDT bypass pin from the default name to “my_bypass” and specifies in the same command that the bypass signal is inverted between the chip input pin and the EDT logic. Afterward, the example again reports the EDT pins.

report_edt_pins

```
//  
//      Pin description          Pin name           Inversion  
//      -----                  -----  
//      Clock                  edt_clock           -  
//      Update                 edt_update           -  
//      Bypass mode            edt_bypass           -  
//      Scan channel 1 input   edt_channels_in1    -  
//      "         " output     edt_channels_out1   -  
//      Scan channel 2 input   edt_channels_in2    -  
//      "         " output     edt_channels_out2   -  
//
```

```
set_edt_pins clock a1  
set_edt_pins bypass my_bypass -inv  
report_edt_pins
```

```
//  
//      Pin description          Pin name           Inversion  
//      -----                  -----  
//      Clock                  a1                  -  
//      Update                 edt_update           -  
//      Bypass mode            my_bypass           inv  
//      Scan channel 1 input   edt_channels_in1    -  
//      "         " output     edt_channels_out1   -  
//      Scan channel 2 input   edt_channels_in2    -  
//      "         " output     edt_channels_out2   -  
//
```

Example 2

The following example reports each pin name and lists all of the blocks that share that pin. Channel sharing information is shown in red font.

report_edt_pins -all -group_by_pin_name

Pin name	Block	Pin Description	Inversion	Channel pipeline stages (mask register)
<hr/>				
edt_clock	A	Clock	-	-
	B	Clock	-	-
	C	Clock	-	-
edt_update	A	Update	-	-
	B	Update	-	-
	C	Update	-	-
channel_in_A	A	Scan channel 1 input (control)	-	7
channel_in_B	B	Scan channel 1 input (control)	-	"
channel_in_C	C	Scan channel 1 input (control)	-	"
channel_in1	A	Scan channel 2 input	-	7
	B	Scan channel 2 input	-	"
	C	Scan channel 2 input	-	"
channel_in2	A	Scan channel 3 input	-	7
	B	Scan channel 3 input	-	"
	C	Scan channel 3 input	-	"
channel_outA	A	Scan channel 1 Output	-	0
channel_outB	B	Scan channel 1 Output	-	"
channel_outC	C	Scan channel 1 Output	-	"

Related Topics

[report_drc_rules](#)
[report_edt_configurations](#)
[report_scan_volume](#)
[set_edt_options](#)
[set_edt_pins](#)
[write_edt_files](#)

report_environment

Context: unspecified, all contexts

Mode: all modes

Displays the current value(s) of many of the most common “set_...” commands.

Usage

```
report_environment [{> | >>} file_pathname]
```

Description

Displays the current value(s) of many of the most common “set_...” commands.

When you first invoke the tool, the values displayed are the default settings for these commands.

This command shows the [set_gate_report](#) command settings in a field titled “gate report”. The tool reports global on/off options only when the options are enabled. The global on options are reportd in a semi-colon separated list in parenthesis. The tool reports clock_cone clock names in single quotes. The tool only reports the value of time values and not the units. The -max_fanouts and -clock_domains_in_capture_cycles options of set_gate_report do not impact the report_environment output.

Arguments

- *>file_pathname*

An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- *>>file_pathname*

An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

The following example reports the current conditions under which the tool tests the circuit, then performs an ATPG run:

```
set_system_mode analysis
add_faults -all
report_environment
create_patterns
```

The output from the report_environment command may look like the following:

```
abort limit = 30
atpg compression = OFF
atpg limits = none
bist initialization = X
bus simulation method = global
capture clock = none
checkpoint = OFF
clockpo patterns = ON
clock restriction = clock_po
clock_off simulation = OFF
contention check = ON, mode = bus, handling = warning
DRC transient detection = ON -Verbose
fails report = OFF
fault mode = uncollapsed
fault type = stuck
gate level = design
gate report = normal
iddq checks = none, handling = warning
iddq strobe = label
learn reporting = OFF
logfile handling = OFF
net dominance = wired-gate
net resolution = wired-gate
observation point = master
pattern classification = ON
pattern source = internal
possible credit = 50%
pulse generators = ON
RAM initialization = uninitialized
RAM test mode = static_pass_thru
random atpg = ON
random clocks = none
random patterns = 1024
screen display = ON
shadow checking = ON
simulation mode = combinational      depth = 0
skew load = OFF
split capture cycle = OFF
stability check = ON
system mode = setup
tester cycles = PLL_capture 10 timeplate_1
TLA loop handling = OFF
trace report = OFF
Z handling = int=X ext=X
Zhold behavior = ON
```

Related Topics

[report_pattern_filtering](#)

report_external_simulator

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the shell command the tool will use to invoke an external simulator for verifying patterns.

Usage

```
report_external_simulator [<> | >>] file_pathname]
```

Description

Displays the shell command the tool will use to invoke an external simulator for verifying patterns.

The report_external_simulator command displays the current setting of the set_external_simulator command. You use the latter to specify the shell command the tool should use to invoke an external time-based simulator for verifying patterns.

Arguments

- `>file_pathname`
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- `>>file_pathname`
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example checks the current external simulator setting, then specifies for the tool to invoke ModelSim in batch mode (-c) with loading messages disabled (-quiet), when verifying simulation results:

```
report_external_simulator  
.../vsim -c  
  
set_external_simulator vsim -c -quiet  
report_external_simulator  
.../vsim -c -quiet
```

Related Topics

[set_external_simulator](#)

report_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis,
patterns -failure_mapping

Mode: analysis

Simulates the current design and test patterns and reports the failures in a pattern-based format.

Usage

```
report_failures [{pin_pathname -Stuck_at {0 | 1}}...] [-Pdet | -Exact] [-Max integer] [-Cycle]  
[-MEasure_scan_at {CHANnels | CHAIns}]  
[{> | >>} file_pathname]
```

For “patterns -failure_mapping”

```
report_failures -UNMapped
```

Description

Simulates the current design and test patterns and reports the failures in a pattern-based format.

For failure mapping, reports the unmapped top-level failures.

Arguments

- *pin_pathname* -Stuck_at {0 | 1}

An optional, repeatable string, switch and literal triplet that specifies both the location and the value of a stuck-at fault to check for failing patterns. A fault simulation is run on the specified fault and the results are reported. Argument options include:

pin_pathname — A string that specifies the pin pathname of the fault whose failing patterns you want to identify.

-Stuck_at 0 | 1 — A switch and literal that specify the stuck-at fault value to simulate. The stuck-at options include:

0 — stuck-at-0

1 — stuck-at-1

- -Pdet

An optional switch that directs the tool to report a failure when an X is simulated where a binary value is expected. By default, only the binary detections are reported.

- -Exact

An optional switch directs the tool to report a failure when an X is simulated where a binary value is expected (like the -Pdet switch) and also when a binary value is simulated where an X is expected. By default, only the binary detections are reported.

- -Max *integer*

An optional switch and integer pair that specifies a maximum number of failing patterns before the command stops the simulation. By default, all failing patterns are simulated.

- **-CYClE**
An optional switch that reports the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the “[Input File Requirements](#)” chapter in the *Tessent Diagnosis User’s Manual*.
- **-measure_scan_at {channels | chains}**
An optional switch and literal pair that instructs the tool to report the failing bits of an EDT-enabled design at the scan elements; normally, the tool reports the failing bits at the EDT external channels. For an EDT-enabled design, this switch is only valid for external pattern sets in ASCII or binary format; STIL and WGL formats are not supported.

If EDT is enabled (On), by default, the tool measures at the external channels so that all failing bits are compressed. If you specify the chains option in this case, the tool reports the failing bits at the scan chain cell_number without considering the output compaction logic.

If EDT is disabled (Off), the tool always measures at the scan cells. In this case, specifying the channels option creates an error condition.

Reporting failures using the chains option generates the following information:

`pattern_id` — pattern index where the failure occurs.
`chain/PO_name` — scan chain or the primary output name where the failure occurs.
`cell_number` — scan cell index where the failure occurs. This field will be empty if the failure occurs at a primary output.
`expected_value` — expected value observed at the scan chain output pin from the external pattern set for the failing point.
`simulated_value` — the simulated value from the internal simulation for the failing point.
`cell_path_name` — cell or primary output name of the failing point.

- **>file_pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of `file_pathname`.
- **>>file_pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of `file_pathname`.
- **-unmapped**
A required switch that specifies to report all failures that Tessent Diagnosis could not map to core instances.

Examples

Example 1

The following example loads a test pattern source, runs simulation, and displays the pattern-based failure information.

```

set_system_mode analysis
read_patterns file1
report_failures

// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec

```

The pattern-based failure data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name as shown in the following example.

```

chain_test
0 chain5 0 H L // uUART/reg_MODE0_CLK
0 chain5 1 H L // uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5 4 H L // uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5 5 H L // uINTR/ix546
0 chain5 8 H L // uINTR/ix476
0 chain5 9 H L // uINTR/ix446
0 chain5 12 H L // uINTR/ix356
0 chain5 13 H L // uINTR/ix326
0 chain5 16 H L // uINTR/ix236
0 chain5 88 H L // padding_cycle
0 chain5 100 H L // padding_cycle
last_pattern_tested 0
scan_test
0 chain5 2 H L // uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5 4 H L // uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5 6 H L // uINTR/ix106
0 chain5 8 H L // uINTR/ix476
0 chain5 9 H L // uINTR/ix446
0 chain5 11 H L // uINTR/ix386
0 chain5 12 H L // uINTR/ix356
0 chain5 13 H L // uINTR/ix326
0 chain5 28 H L // uSTM/reg_WAIT_SYNC/SFFR
0 chain5 100 H L // padding_cycle
last_pattern_tested 2

```

Example 2

The following example shows the output of the report_failures command when the “-measure_scan_at chains” switch and option combination is specified for an EDT-enabled design:

```

FAULT> read_patterns patterns.ascii
FAULT> report_failures -measure_scan_at chains

// pattern_id chain/    cell    expected simulated cell_path_name
//          PO_name   number  value      value
0           chain1    2        H        L          // all_rams_i/sf_0003_1
0           chain4    3        H        L          // all_rams_i/sf_0028_1
0           chain10   1        L        H          // all_rams_i/lf_0055_1
0           chain11   5        L        H          // all_rams_i/lf_0043_1
1           chain7    4        H        L          // all_rams_i/sf_0053_1

// incorrect_patterns=2, simulated_patterns=61, simulation_time=0.04sec.

```

Example 3

The following example returns a list of the top-level failures for core instance logic_die_BOT_inst_logic_die_internal.

report_failures -unmapped

```
//Note: TestSuite[1] '../data/top_flog1', #fail_bits=1Cycle Pin  
      Expected Actual688 bc_2_core_2_q1 L H
```

Related Topics

[diagnose_failures](#)

[read_failures](#)

[read_patterns](#)

[write_failures](#)

report_false_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays false paths previously defined with the [add_false_paths](#) command.

Usage

```
report_false_paths [-All] [-Error] [-Verbose] [-Debug_error]
  [-From source_node... | -FROM_Clock clock_signal_name...]
  [-TO sink_node... | -TO_Clock clock_signal_name...]
  [{-THrough through_node...}...]
  | [-GATE {{gate_ID# | pin_pathname}...}] ...
  | [-X_Statistics [-PATH_Count {int | All}] [-PATH_X_threshold int]
    [-TO_GATE_Count {int | ALL}] [-TO_GATE_X_threshold int]]
  [{> | >>} file_pathname]
```

Description

Displays false paths previously defined with the [add_false_paths](#) command.

Note that you can display SDC filenames and line numbers by first issuing the “[set_timing_exceptions_handling](#) -line_no” command. To analyze if a pin or a gate is affected by a false path, you can use “[set_gate_report](#) -timing_exceptions on” and [report_gates](#) commands.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) false paths are reported.

Arguments

When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. An asterisk is treated as a wildcard character, enabling you to use it to match many pathnames in the design.

- **-All**
An optional switch that displays all previously-defined false paths. This is the default.
- **-Error**
An optional switch that displays only the false paths that have errors and warnings.
- **-Verbose**
An optional switch that displays all the gate information in the from, to, and through gate lists.
- **-Debug_error**
An optional switch that displays all the previously-defined false paths that have errors and warnings, along with their respective error and warning messages.

- -From *source_node*...

An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, all the output pins of the instance are considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

- -FROM_Clock *clock_signal_name*...

An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.

- -TO *sink_node*...

An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

Note

 For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_false_paths... -to CLK2” would report false paths with an end point at D2.

- -TO_Clock *clock_signal_name*...

An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal, and the data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.

- {-THrough *through_node*...}...

An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, their order, left-to-right, in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

```
add_false_paths -from A -through B C -through D E -to F
```

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be false paths.

- **-GAt gate-ID# | pin_pathname**

An optional repeatable switch and repeatable integer or string that filters the report results based on gates or pins that you specify. The tool reports only the false paths and multicycle paths that include the specified gates or pins as the definition points (-from, -to, or -through). You can use multiple -Gate switches to specify additional pins or gates in a single command invocation.

gate-ID# — A repeatable integer that specifies the gate identification number. When you specify multiple gate IDs, the command reports on any path that includes any of those gates as definition points.

pin_pathname — A repeatable string that specifies the name of a pin within the design. When you specify multiple pin_pathnames, the command reports on any path that includes any of those pins as definition points.

- **-X_Statistics [-PATH_Count int | All] [-PATH_X_threshold int] [-TO_GATE_Count {int | ALL}] [-TO_GATE_X_threshold int]**

An optional switch that reports the false or multicycle paths that have the most X masking from [create_patterns](#) or [simulate_patterns](#) after you enable the collection of X statistics with the [set_timing_exceptions_handling](#) command. The command reports the full path only when the path is not associated with the SDC file. Otherwise, the command reports only the SDC filename and line number of the path, as well as the number of cumulative Xs.

-PATH_Count {int | ALL} — An optional switch and integer or literal pair that specifies the number of paths to be reported. The default is to report up to 10 false or multicycle paths that have the greatest number of masking effects propagated to some to-points of the timing exception paths.

int — An optional positive integer that specifies the number of paths for which the tool reports X statistics.

ALL — An optional literal that specifies the tool reports X statistics for all paths that have X statistics.

-PATH_X_threshold int — An optional switch and positive integer pair that specifies the minimum number of accumulated Xs needed for the command to report the path.

-TO_GATE_Count {int | ALL} — An optional switch and integer or literal pair that specifies that the tool reports some or all of the to-gates' X statistics, immediately following each false path, in order from the gate with the greatest number of maskings to the least. The to-gates are the false path defined points from using the -to

option. When the -to option is not specified, path to-gates are derived from the last -through option or the -from option, when the -through option is not specified.

int — An optional positive integer that specifies the number of to-gates for which the tool reports X statistics.

ALL — An optional switch that specifies the tool reports all to-gates with at least one masking.

-TO_GATE_X_threshold *int* — An optional switch and integer pair that specifies the minimum number of accumulated X's needed for the command to report to-gates. When the **-to_gate_x_threshold** option is specified, and the **-to_gate_count** is not specified, the default is 10 gates.

By default, if the **-to_gate_count** option and the **-to_gate_x_threshold** option are both unspecified, the tool does not report any to-gates.

The **-to_gate_count** and the **-to_gate_x_threshold** reporting criteria will be the AND operation of both options. For example, if “**-to_gate_count 5**” and “**-to_gate_x_threshold 8**” are specified, the tool reports the 5 to-gates that have at least 8 accumulated Xs.

- **> *file_pathname***
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays all previously-defined false paths.

```
report_false_paths -all
```

```
// False Path -from LINESTATE[1] LINESTATE[0] -to_clock MCLK
// False Path -from_clock UTMICLK[0] -to_clock MCLK
//
// -----
// The false path below has an error.
// -----
// False Path -from dummy1 -to dummy2
//
// -----
// The false paths below have warnings.
// -----
// False Path -from_clock S -to_clock s
// False Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN
// False Path -from CLK1 -through U7/CP CLK1
//
// Total paths reported = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//           Use the command "report_false_paths -debug_error" to report
//           the causes of the errors/warnings.
// Warning: False path defined through all primary inputs, including all
//           clocks, which prohibits at-speed testing. Deleted clocks from
//           this list.
// Warning: There exists a false path that prohibits at-speed testing of
//           clock domain /CLK1.
//           This may result in lower test coverage.
```

Example 2

The following example displays the false paths with error and warning messages.

report_false_paths -error

```
// Error: False Path -from dummy1 -to dummy2
// Warning: False Path -from_clock S -to_clock s
// Warning: False Path -from CLK1 -through U7/CP CLK1
// Warning: False path -through I1 I2 I3 I4 I5 SIN CLK1 SEN
// Total reported false paths = 4
```

Example 3

The following example displays debug information for the false paths with error and warning messages.

report_false_paths -debug_error

```
// Error: False Path -from dummy1 -to dummy2 did not find gates dummy1
// dummy2.
// Warning: False Path -from_clock S -to_clock s has no combinational path
//           between start and end points.
// Warning: False Path -from CLK1 -through U7/CP CLK1 prohibits at-speed
//           testing of clock domain /CLK1.
// Warning: False path -through I1 I2 I3 I4 I5 SIN CLK1 SEN defined
//           through all primary inputs, including all clocks, which
//           prohibits at-speed testing. Deleted clocks from this list.
// False path debug summary: 1 error, 3 warnings
```

Example 4

The following example displays the false paths that start at /my_design/d_in[6]. The second path defined by the user results in an error because there is no sensitized path between the gates. Using the -debug_error option provides additional information about the error.

```
report_false_paths -verbose -from /my_design/d_in[6]

// False path -from /my_design/d_in[6] ( /reg_d_1_7/Q (81)
//      /reg_d_1_6/Q (73) ) -to /my_design/u25 ( /my_design/u25/Y (130) )
//
// -----
// The false path below has an error.
// -----
// Error: False path -from /my_design/d_in[6] ( ) -to /my_design/xray[5] ( )
//
// Total reported false paths = 2
// Warning: The last 1 path reported has an error.
//           Use the command "report_false_paths -debug_error" to report
//           the cause of the error.

report_false_paths -verbose -from /my_design/d_in[6] -debug_error

// Error: False Path -from /my_design/d_in[6] -to /my_design/xray[5] \
//         has no sensitizable combinational path between start and end points.
// False path debug summary: 1 errors, 0 warnings
```

Example 5

The following example displays the false paths and multicycle paths whose definition points (-from, -to, or -through) include the gate top/inst1/A:

```
report_false_paths -gate top/inst1/A
```

Example 6

The following shows false path reporting after turning on reporting of SDC filenames and line numbers:

```
read_sdc timing.sdc
set_timing_exceptions_handling -line_no on
report_false_paths

// False Path -from_clock cmem_fclk1 cmem_fclk1n -to_clock p_clk
//     spm_pci_clk -setup (filename = timing.sdc, line 406)
// False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclk1
//     cmem_fclk1n -setup (filename = timing.sdc, line 408)
...
// Total reported false paths = 177
```

Example 7

The following shows the commands to enable X statistics functionality and create ATPG patterns. It reports the 3 false paths with the greatest masking effect, in order starting with the path that has the most Xs. For each path, the tool reports the X statistics for the top 5 top-gates, which observe the most masking, in order starting from the gate that has the most Xs.

```

read_sdc mod.sdc
set_timing_exceptions_handling -x_statistics on
create_patterns
report_false_paths -x_statistics -path_count 3 -to_gate_count 5

// Total 249 paths out of 350 paths produced Xs.
// -----
// Path Type Number of Xs File and line number
// -----
// 1 false path 1572811 mod.sdc (line 32761)
// -----
// -----
// Gate (Id) Number of Xs
// -----
// '/u_lspipe/dcuv_valid_dc3_reg/' (253337) 3162
// '/clk_gate_list_phys_id_q_reg_0_/' (218650) 2656
// '/clk_gate_list_phys_id_q_reg_3_/' (218653) 2684
// '/clk_gate_list_phys_id_q_reg_2_/' (218652) 1886
// '/clk_gate_list_phys_id_q_reg_1_/' (218651) 1006
// Only the first 5 to-gates with most masking are reported (21524 out
// of total 26000 to-gates captured X).
// -----
// 2 false path 1107367 mod.sdc (line 32761)
// -----
// -----
// Gate (Id) Number of Xs
// -----
// '/clk_gate_list_phys_id_q_reg_0_/' (218650) 2656
// '/clk_gate_list_phys_id_q_reg_1_/' (218651) 2312
// '/clk_gate_list_phys_id_q_reg_3_/' (218653) 2110
// '/clk_gate_list_phys_id_q_reg_5_/' (218655) 1118
// '/clk_gate_list_phys_id_q_reg_2_/' (218652) 1092
// Only the first 5 to-gates with most masking are reported (14547 out
// of 16000 to-gates captured X).
// -----
// 3 false path 113454 mod.sdc (line 30390)
// -----
// -----
// Gate (Id) Number of Xs
// -----
// '/u_coreoverpowerdown/cluster_id_reg_reg_3_/' (134183) 779
// '/fwd_data_sync_reg_40_/' (170295) 654
// '/clken_reg/' (230954) 548
// 'gen_resync_0_gen_resync_r_u_resync/' (180823) 545
// '/state_reg_0_/' (238777) 520
// Only the first 5 to-gates with most masking are reported (1359 out
// of 1800 to-gates captured X).
// -----
// Only the first 3 paths producing the most Xs were reported.

```

Related Topics

- [add_false_paths](#)
- [delete_false_paths](#)
- [delete_multicycle_paths](#)

[read_sdc](#)

[report_multicycle_paths](#)

[set_timing_exceptions_handling](#)

report_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.

Usage

Path Delay Faults Usage:

```
report_fault_sites [-All | path_name] [-Path gate_id_begin gate_id_end]
[-Display {[FLAt_schematic] [HIEarchical_schematic] [DAta] | ALl} [APPend]]
[> | >>} file_pathname]
```

Bridge Faults Usage:

```
report_fault_sites
{[-All | net_pathname_pair | bridge_name... | net_pathname... | instance_name...]
[-NAME | -SINGle] [-NET_pathname] [-PIN_pathname] [-ATtribute] [-NOEQ] | 
[-Histogram {Layer | Type | {Weight [-Sets number]} }]}
[-Display {[FLAt_schematic] [HIEarchical_schematic] [DAta] | ALl} [APPend]]
[> | >>} file_pathname]
```

UDFM Usage:

```
report_fault_sites {[-ALl | instance_name] [-Udfm_type name][-CELL name]
[-MOdule name] [-FAult name]} |
{[-All | instance_name] [-UNDEFined_cells] | [-DEFined_cells]}
[> | >>} file_pathname
```

Description

Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.

The report_fault_sites command displays the definitions in the tool's internal list for the specified faults. Fault definitions are loaded into the internal list with the [read_fault_sites](#) command.

For more information about bridge faults, refer to “[The Static Bridge Fault Model](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information about path delay faults, refer to “[Path Delay Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information about UDFM faults, refer to “[About User-Defined Fault Modeling](#)” in the *Tessent Scan and ATPG User’s Manual*.

Path Delay Specifics

The report_fault_sites command also displays unambiguous paths you previously added to the internal path list using the add_ambiguous_paths command. When displaying derived unambiguous paths, report_fault_sites cannot list a gate if it does not have a pin pathname associated with it. For some designs, gates without associated pin pathnames could be fairly common. Therefore, refer to the displayed unambiguous paths for launch and capture points, but be aware the display may not show all the gates in between.

Note

 If you get ambiguous path warnings, then add_ambiguous_paths, those paths when written out lose the information about which of the ambiguous paths they were. Consequently, you cannot reload them and get back to the same set of paths.

UDFM Specifics

The report_fault_sites command doesn't report restricted information about UDFM models generated by Tesson CellModelGen, which protects the IP inside of those models.

Arguments

- **-All**
An optional switch that displays all fault definitions for the currently loaded paths, bridge, or UDFM entries. This is the default.
- ***path_name***
An optional string that specifies the name of a path whose definition you want to display. The path must be one defined in a path definition file that you previously loaded using the read_fault_sites command.
- **-Path *gate_id_begin* *gate_id_end***
An optional switch and two-integer triplet that specifies a particular path or portion of a path whose definition you want to display. Use this argument to report on paths that were not defined in a path definition file, and therefore were not loaded using the read_fault_sites command.
The two integers specify two gate identification numbers that indicate the beginning and ending of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.
The value of *gate_id_begin* or *gate_id_end* is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.
- ***net_pathname_pair***
A string pair that specifies the pathname to the nets or pins in a bridge entry. The first string specifies NET1, and the second string specifies NET2.
- ***bridge_name***
A repeatable string that specifies a bridge name.

- ***net.pathname***
A repeatable string that specifies a single net or pin pathname.
- ***instance.name***
A repeatable string that specifies an instance pathname.
- **-ATtribute**
An optional switch that displays the attributes of the reported bridge entries. Attributes may include distance, weight, layer, and parallel_run.
- **-NAME**
An optional switch that specifies all entered strings are bridge names.
- **-SINGle**
An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.
If the string is an instance that contains a non-dominant net from a bridge entry, a fault is reported for each bridge entry associated with the instance.
- **-NET.pathname**
An optional switch that reports net pathnames for bridge entries. This is the default.
- **-PIN.pathname**
An optional switch that reports pin pathnames for bridge entries instead of net pathnames.
- **-NOEQ**
An optional switch that reports only the representative fault information: the net pair and fault categories.
- **-Histogram {Layer | Type | {Weight [-Sets *number*]}}**
An optional switch and literal pair that displays a histogram of the number of equivalent classes of bridges based on either their layer, type, or weight attribute. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying “-sets 0” forces the tool to display an individual line for each bridge of different weight.
The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.
- **-Display {[FLAt_schematic] [HIErarchical_schematic] [DAta] | ALI} [APpend]**
An optional switch and literal that displays the reported information graphically in the specified DFTVisualizer window(s); if *Append* is specified, the information is added to the existing contents of the window instead of replacing the contents. The choices are as follows:
 - FLAt_schematic — Flat Schematic window
 - HIErarchical_schematic — Hierarchical Schematic window

DAta— Data window

ALl — A literal that displays the information in all of the preceding windows.

- **-Udfm_type *name***

An optional switch and string pair that specifies the name of a UDFM fault type. Use this option to display the faults associated with the specified UDFM fault type. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-Cell *name***

An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-Module *name***

An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with the specified module. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-Fault *name***

An optional switch and string pair that specifies the name of a UDFM defined fault. Use this option to display all faults associated with the specified name. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-UNDEFined_cells**

An optional switch that reports cells without a valid UDFM definition that were added with a previous [read_fault_sites](#) or [add_fault_sites](#) command.

- **-DEFined_cells**

An optional switch that reports cells with a valid UDFM definition that were added with a previous [read_fault_sites](#) or [add_fault_sites](#) command.

- **> *file.pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file.pathname*.

- **>> *file.pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

Example 1

The following path delay example reads in (loads) the path information and then displays that data:

```
set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites
```

```

PATH "path0" =
  PIN /I$6/Q + ;
  PIN /I$35/B0 + ;
  PIN /I$35/C0 + ;
  PIN /I$1/I$650/IN + ;
  PIN /I$1/I$650/OUT - ;
  PIN /A_EQ_B + ;
END ;
PATH "path1" =
  PIN /I$6/Q + ;
  PIN /I$35/B1 + ;
  PIN /I$35/C1 + ;
  PIN /I$1/I$649/IN + ;
  PIN /I$1/I$649/OUT - ;
  PIN /I$5/D - ;
END ;

```

Example 2

The following bridge example reads the bridge information in the bridge fault definition file, */user/design/bridgefile*, then displays the data for bridge entries added to the fault site list:

```

set_fault_type bridge read_fault_sites /user/design/bridgefile

// Load bridge entries from file "/user/design/bridgefile".
//
// 5 bridge entries were read from file "/user/design/bridgefile".
// 2 bridge entries were skipped due to unknown net name.
// 3 bridge entries were added to the fault site list.

```

report_fault_sites

```

BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U0.BRIDGE";
}

```

Example 3

The following example displays the attribute data for the “U0.BRIDGE” bridge entry:

```
report_fault_sites U0.BRIDGE -name -attribute
```

```
BRIDGE {
    NET1 = "/G11";
    NET2 = "/G10";
    FAULTS = {-, -, -, -, -};
    WEIGHT = 1.0000;
    DISTANCE = 4.93827000e-06 um;
    PARALLEL_run = 6.07500000e-05 um;
    X_COORDINATE = 152500;
    Y_COORDINATE = 739500;
    LAYER = "metall1";
    NAME = "U0.BRIDGE";
}
```

Example 4

The following example displays the UDFM fault definition f01 defined in UDFM type my-udfm-sa-faults of instance /top/inst1:

```
report_fault_sites /top/inst1 -udfmtype my-udfm-sa-faults -fault f01

UDFM {
    Version:1;
    UdfmType("my-udfm-sa-faults") {
        Instance("/top/inst1") {
            // Source reference:
            // File: /home/JohnDoe/myadded.udfm, Line: 145
            Fault("f01") {
                Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
                Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
            }
        }
    }
}
```

If there is more than one UDFM types used, the following command reports all f01 faults on instance /top/inst1:

```
report_fault_sites /top/inst1 -Fault f01
```

```
UDFM {
    Version:1;
    UdfmType("my-udfm-sa-faults") {
        Instance("/top/inst1") {
            // Source reference:
            // File: /home/JohnDoe/myadded.udfm, Line: 145
            Fault("f01") {
                Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
                Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
            }
        }
    }
    UDFMType("intra-cell") {
        Instance("/top/inst1") {
            // Source reference:
            // File: /home/JohnDoe/myadded.udfm, Line: 418
            Fault("f01") {
                Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
                Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
            }
        }
    }
}
```

report_fault_sites -all

```
UDFM {
    Version:1;
    UdfmType("my-udfm-sa-faults") {
        Cell("AOX") {
            // Source reference:
            // File: /home/JohnDoe/myadded.udfm, Line: 145
            Fault("f01") {
                Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
                Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
            }
        }
    }
}
```

Example 5

The following example shows how to display a list of cells both with and without valid UDFM definitions:

report_fault_sites -defined_cells

```
// Report of cells defined by UDFM
//
// Cell Name Instances
// -----
// XNOR2X18      518
// XNOR2X3       139
// XOR2X17       387
// AND2X35       913
// AO112X27        4
// AO12X27      203
// AO22X27      219
// AO312X27        39
// AO31X27         5
// -----
```

report_fault_sites -undefined_cells

```
// Report of cells not defined by UDFM
//
// Cell Name Instances
// -----
// cnhlsx4v      52
// df1qx1v      834
// df4sqx1v        7
// u_noti       68
// u_mux2      7261
// u_mux3       20
// u_mux4      398
// -----
```

Related Topics

[add_ambiguous_paths](#)
[add_fault_sites](#)
[add_faults](#)
[delete_fault_sites](#)
[read_fault_sites](#)
[set_fault_type](#)
[write_fault_sites](#)

report_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Displays fault information from the current fault list.

Usage

Path Delay Usage:

```
report_faults [-Both | -Rise | -Fall] [-VERilog | -VHdl]
[-Class fault_class[fault_subclass]] [object_pathname] [-NO_Subclass]
```

Bridge Usage:

```
report_faults {[-Histogram Weight [-Sets number]]}
| {[-Histogram {Layer | Type}]}
| {net_pathname_pair | {bridge_name ... -NAME}
| {net_pathname ... -SINGLe} | {instance ... -SINGLe}}
[-Class fault_class[fault_subclass]] [-NET_pathname | -PIN_pathname] [-NOEQ]
[-NO_Subclass] [-VERilog | -VHdl] [-ATtribute] [-SCAN_Enable] [-CLOCK_Cones]
[-IO] [-ASYnchronous_controls}]
| {[-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUIvalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
[-CAPture_procedures {ALL | capture_procedure_name ...}]} [(> | >>) file_name]
```

Toggle/Iddq Usage:

```
report_faults [-Stuck_at [01 | 0 | 1]] [-NO_Subclass]
[{-Class fault_class[fault_subclass]} [{pin_pathname | gate_id} ...] [-NOEQ]
[instance_name ...] [-Hierarchy integer] [-Min_count integer] [-VERilog | -VHdl]
[-CELL_name] [-CLOCK_Domains
{ALL | clock_pathname ...} [-NO_EQUIvalent_clocks]]
[-CAPture_procedures {ALL | capture_procedure_name ...}]}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
| {-UNlisted}
[-FRom pin...] [-TThrough pin...] [-TO pin...]
[-STOP_at {sequential_elements | scan_cells | ports_only}] [(> | >>) file_name]
```

Stuck/Transition Usage:

```
report_faults [-Stuck_at [01 | 0 | 1]] [-NO_Subclass]
[{-Class fault_class[fault_subclass]} [{pin_pathname | gate_id} ...] [-NOEQ]
[instance_name ...] [-Hierarchy integer] [-Min_count integer] [-VERilog | -VHdl]
[-CELL_name]
[-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUIvalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
[-CAPture_procedures {ALL | capture_procedure_name ...}] [-DElay_data]
[-TIMING_CRITICAL}] | {[-SCAN_Enable] [-CLOCK_Cones] [-IO]
[-ASYnchronous_controls]} | {-UNlisted}
```

```
[-FRom pin...] [-THrough pin...] [-TO pin...]  
[-STOP_at {sequential_elements | scan_cells | ports_only}] [(> | >>) file_name]
```

UDFM Usage:

```
report_faults [-NO_Subclass] {[ -Class fault_class[fault_subclass]]  
| instance_pathname | [-Hierarchy integer] [-NOEQ]  
[-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUIvalent_clocks]  
[-EXCLUDEFAULTS_BETWEEN_SYNC_clock_domains] {[ -UDFM_type name]  
[-INstance name] [-CELL name] [-MODule name] [-SOurce]} }  
| {-UNlisted} [-FRom pin ...] [-THrough pin ...] [-TO pin ...]  
[-STOP_at [ sequential_elements | scan_cells | ports_only }] [(> | >>) file_name]
```

Power-Aware Options:

(applicable only after you have loaded CPF/UPF power data)

```
[[ON_domains] | [-OFF_domains] | [-POWER_domains {domain_name ...}]]  
[-ISolation_cells] [-LLevel_shifters] [-REtention_cells]
```

Description

Displays fault information from the current fault list. The report_faults command displays all the faults you added to the fault list using the add_faults or read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any arguments, report_faults displays information on all the known faults.

By default, the report_faults command ignores the constraint values implied in the data path when reporting faults in the data path. However, the tool still considers the constraint values for clock path faults.

For more information about UDFM faults, refer to “[About User-Defined Fault Modeling](#)” in the *Tessent Scan and ATPG User’s Manual*.

The power-aware options report_faults based on power domains and/or power features after you have loaded a CPF or UPF file.

The report_faults command displays the following columns of information for each fault:

- fault value - The fault value may be either 0 (for stuck-at-0 or “slow-to-rise” transition faults) or 1 (for stuck-at-1 or “slow-to-fall” transition faults).
- fault code - A code name that indicates the lowest level fault class assigned to the fault (see [Table 5-10](#)). This also includes a fault sub-class code when available (see [Fault Sub-classes](#) in the *Tessent Scan and ATPG User’s Manual*).
- fault site - The pin pathname of the fault site.
- cell name (optional) - The name of the cell corresponding to the fault (displayed only if you include the -Cell_name argument).

You can use the -Hierarchy option to display a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option which specifies the minimum number of faults that must be in a hierarchical level before they are displayed.

You may choose to display either collapsed or uncollapsed faults by using the set_fault_mode command. Also, some fault data is large and it would be more appropriate to use the write_faults command, and then read the file contents.

Arguments

- -Class *fault_class*[.*fault_subclass*]

An optional switch and literal pair that specifies the fault class or fault class/sub-class to display. [Table 5-10](#) lists the valid fault class codes and their associated fault class names. [Fault Sub-classes](#) in the *Tessent Scan and ATPG User's Manual* lists the valid fault sub-class names. You should use the code when specifying a fault class or sub-class.

When you have previously issued the “[report_statistics -detailed_analysis](#)” command, report_faults can report specific AU.TC and AU.PC faults by pin pathname or by gate ID. For more information, refer to “[Example 9](#)” on page 1566.

Table 5-10. Fault Class Codes and Names

Fault Class Code	Fault Class Name	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE

Table 5-10. Fault Class Codes and Names (cont.)

Fault Class Code	Fault Class Name	Fault Class Coverage
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- **-NO_Subclass**
Prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when the tool has analyzed them.
- **-UNlisted**
Displays unlisted fault data. If there are multiple grayboxed instances in the design, this switch reports accumulated fault numbers.
- **-FRom pin ...**
This option sets the start pin(s) of the user-specified cone to report the faults. It directs the tool to perform forward cone tracing from the specified pin(s) until it encounters a stop point. You can specify one or more instance names, rather than pin names. When you specify an instance name with -from, its output pins are used when defining the cone.
- **-THrough pin ...**
This option sets the through pin(s) of the cone of logic used to select faults to report. When specified, the tool performs forward and backward cone tracing from the specified pin(s) until it encounters a stop point in both directions. When you specify an instance name with -through, its output pins are used when defining the cone.
- **-TO pin ...**
This option sets the end pin(s) of the cone of logic used to select faults to report. When specified, the tool performs backward cone tracing from the specified pin(s) until it encounters a stop point. When you specify an instance name with -to, its input pins are used when defining the cone.

Note



When -from, -to, and -through are used with other switches, the selected faults will be the intersection of the cones traced for all specified switches.

- **-STOP_at {sequential_elements | scan_cells | ports_only}**
This option sets the stop condition for tracing of the cone of logic used to select faults to report. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify “scan_cells”, the cone includes non-scan cells and tracing stops only at scan cells. When you specify “ports_only”, the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

- **-Stuck_at 01 | 0 | 1**

An optional switch and literal pair that specifies the stuck-at or transition faults you want to display. The choices are as follows:

01 — A literal specifying that for stuck-at faults the tool display both “stuck-at-0” and “stuck-at-1” faults; or for transition faults, the tool display both “slow-to-rise” and “slow-to-fall” faults. This is the default.

0 — A literal specifying to display only the “stuck-at-0” faults (“slow-to-rise” faults for transition faults).

1 — A literal specifying to display only the “stuck-at-1” faults (“slow-to-fall” faults for transition faults).

- **object_pathname**

An optional repeatable string that specifies a list of pins, instances, or delay paths whose faults you want to display.

- **-Hierarchy integer**

An optional switch and integer pair that specifies the maximum hierarchy level for which you want to display a summary of the faults. However, note that the reported number of faults depends on other switches, like -class.

- **-Min_count integer**

An optional switch and integer pair that you can use with the -Hierarchy option to specify the minimum number of faults that must be in a hierarchical level to display the hierarchical summary. The default is 1.

- **-NOEQ**

- For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that turns off the display of “EQ” as the fault class for any equivalent faults; the fault class displayed is then that of the representative fault. When you do not specify this switch, the tool displays an “EQ” as the fault class for any equivalent faults. This switch is meaningful only when the [set_fault_mode](#) command is set to Uncollapsed. For more information about representative and equivalent faults, see “[Fault Collapsing](#)” in the *Tessent Scan and ATPG User’s Manual*.

- For bridge faults only, an optional switch that writes only the representative bridge fault information: the net pair and fault classes of each corresponding dominant net. The tool does not report physical data.

- **-Both| -Rise | -Fall**

An optional switch that specifies which faults to display for each path already added via the [read_fault_sites](#) command. These switches are used for path delay faults only.

-Both - An optional switch that specifies to display both the slow-to-rise and slow-to-fall faults. This is the default.

-Rise - An optional switch that specifies to display only the slow-to-rise faults.

-Fall - An optional switch that specifies to display only the slow-to-fall faults.

- **-VERilog | -VHdl**

An optional switch pair that output the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist independent format.

- **-CELL_name**

An optional switch that lists the name of the corresponding circuit element (Tessent Cell library cell, Verilog primitive, primary input or primary output) for each fault as listed in [Table 5-11](#). Cell names for equivalent faults are based on the cells associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 5-11. Name Conventions Used by report_faults -Cell_name

Corresponding Circuit Element	Listed Cell Name
Tessent Cell library cell	Name of the Tessent Cell library cell
Supported Verilog primitive (for a list, see Table 4-1 in the <i>Tessent Cell Library Manual</i>)	Name of the Verilog primitive
Primary input or output	“primary_input” or “primary_output”

- **-CLOCK_Domains {ALL | clock_pathname...}**

A required switch and literal or repeatable string pair that specifies a list of clocks which is used by the tool to decide the faults to be added to the fault list, given the following requirements are met:

- **Static Faults** — A fault is added to the fault list if it can be captured by any clock in the specified list of clocks or any of its equivalent clocks.
- **Transition Faults** — A fault is added to the fault list:
 - i. If the launch and capture clock are the same clock from the specified list of clocks
 - ii. If the launch and capture clock are synchronous clocks from the specified list of clocks
 - iii. If the launch or capture clock from the specified are equivalent equivalent clocks

When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

The argument choices are as follows:

ALL — A literal that specifies all the clocks in the design.

clock_pathname — A repeatable string that specifies a particular clock.

Be aware that a fault reported using this switch might also be detectable by an unspecified clock.

The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when adding or deleting faults by clock domain.

- -NO_EQUIvalent_clocks

An optional switch that prevents the -Clock_domains switch from reporting faults in equivalent clock domains.

- -EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains

An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When specified, only faults within clock domains are considered when adding or deleting faults by clock domain.

- -CAPture_procedures {ALL | *capture_procedure_name...*}

An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to report faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

ALL — A literal that specifies all enabled named capture procedures.

capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

See the [set_capture_procedures](#) command description for information about enabling or disabling named capture procedures.

- -DElay_data

An optional switch that reports the fault list including the slacks of the faulty site. The reported slack is for information only and cannot be loaded back into the system.

- -TIMING_CRITICAL

An optional switch that reports timing-critical faults.

Note

 In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See “[Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual* for complete information.

- -SCAN_Enable

An optional switch that specifies to report faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch reports all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- **-CLOCK_Cones**
An optional switch that deletes all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fan-out of the clock sources. This switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.
- **-IO**
An optional switch that specifies to report faults that are:
 - Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
 - Only observed by POs
- **-ASYnchronous_controls**
An optional switch that specifies to report all faults that only fan out to Set/Reset ports of state elements and RAMs.
- **-Histogram Weight [-Sets *number*] -Histogram {Layer | Type}**
An optional switch and literal pair that displays a histogram of the number of equivalent classes of bridges based on either their layer or type, or their weight attributes. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying “-sets 0” forces the tool to display an individual line for each bridge of different weight.
The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.
- **> *file_name***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_name*.
- **>> *file_name***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_name*.
- ***net_pathname_pair***
An optional, string pair that specifies the bridge fault site. The first string specifies NET1, and the second string specifies NET2.
- ***bridge_name***
An optional, repeatable string that specifies the bridge name.
- ***net_pathname***
An optional, repeatable string that specifies a single net pathname.

- *instance_name...*
An optional, repeatable string that specifies an instance pathname.
- **-ATtribute**
An optional switch that specifies to display the attributes for bridge entries. Attributes may include distance, weight, layer, and parallel_run.
- **-NAME**
An optional switch that specifies all entered strings are bridge names.
- **-SINGle**
An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.
If the string specifies an instance that contains a non-dominant net from a bridge entry, a fault is reported for each bridge entry associated with the instance.
- **-NET.pathname**
An optional switch that reports net pathnames for bridge entries. This is the default.
- **-PIN.pathname**
An optional switch that reports pin pathnames for bridge entries instead of net pathnames.
- *instance_pathname ...*
An optional repeatable string that specifies a list of instance names you want to display.
- **-Udfm_type name**
An optional switch and string that specifies the name of a UDFM fault type. Use this option to display the faults associated with a specified UDFM fault type. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.
- **-CELL name**
An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.
- **-Module name**
An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with a specified module. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.
- **-INSTance name**
An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-S**ource

An optional switch that outputs the UDFM file name and line number where the fault is defined.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- **-ON_domains**

An optional switch that reports faults on all power-on domains.

- **-OFF_domains**

An optional switch that reports faults on all power-off domains.

- **-Power_domains {domain_name ...}**

An optional switch and repeatable string that reports faults on the specified power domains (*domain_name*).

- **-Isolation_cells**

An optional switch that reports faults on all isolation cells.

- **-Level_shifters**

An optional switch that reports faults on all level-shifters.

- **-REtention_cells**

An optional switch that reports faults on all retention cells.

Examples

Example 1

The following example modifies the current fault population in the tool with the contents of the external fault file, *my_detected_faultlist*, then displays the faults:

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge
report_faults
```

Example 2

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y, including the name of the cell associated with the fault:

```
report_faults -stuck_at 0 /mult_8/Y -cell_name
//   type    code    pin.pathname  (cell_name)
//   ----  -----
//     0      DS     /mult_8/Y      (xnor2)
```

Example 3

The following example reports on a fault associated with a primary input:

```
report_faults -stuck_at 1 /ip_scan_en -cell_name
```

```
// type code pin.pathname (cell_name)
// ---- -----
// 0 AU /ip_scan_en primary_input
```

Example 4

The following example shows a fault list with the static test slack and the actual minimum test slack. Only DS faults contain the actual minimum test slack data.

report_faults -delay_data

```
// type code pin.pathname
// ----- -----
// 0 DS /CORE_i/IRQA_B/A 10.1010 12.3000
// 1 DS /CORE_i/IRQA_B/B 20.1480 25.4820
// 0 AU.BB /CORE_i/IRQA_B/C 12.1890 -
// 1 AU.BB /CORE_i/IRQA_B/C 18.8770 -
```

Example 5

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y plus all isolation cells:

report_faults -stuck_at 0 /mult_81/Y -isolation_cells

Example 6

The following example shows the reporting of unlisted faults:

report_faults -unlisted

	detections	collapsed_fault_count	uncollapsed_fault_count
UC	0	1252	2079
UC.EAB	0	452	543
DS	1	69873	87232
DS	2	12873	21432
DS	3	9752	11974
DS	4+	4487	6293
AU.BB	0	8708	10046
AU	0	2374	3782

Example 7

The following example displays the fault data for *ds* class faults associated with library cells that start with *mux* or *an* to the *my_fault_list* file:

report_faults -class ds -cell mux* -cell an*

```
// Command output
```

```
FaultInformation {
    version : 1;
    FaultType (UDFM) {
        FaultList {
            Format : Identifier, Class;
            UdfmType ("intra-cell-bridges_1") {
                Cell ("MUX2") {
                    Instance ("/top/i1/i4") {
                        "F2", DS;
                    }
                }
                Cell ("AND3") {
                    Instance ("/top/i1/i456") {
                        "F2", DS;
                        "F4", DS;
                        "F5", DS;
                        "F7", DS;
                    }
                }
            }
        }
    }
}
```

Example 8

The following example shows the default format of the fault report when the fault type is specified as UDFM using the set_fault_type command:

report_faults

Class	UdfmType	Fault	instance
UC	my-added-sa	f01	/top/i1/i4
DS	my-added_sa	f02	/top/i1/i4
DS	intra-cell	f01	/top/i3
AU.BB	intra-cell	f02	/top/i3

Adding the -source switch to the command extends the report with references to the definition sources:

report_faults -source

Class	UdfmType	Fault	instance	File	Line
UC	my-added-sa	f01	/top/i1/i4	/home/JohnDoe/myadded.udfm	14
DS	my-added_sa	f02	/top/i1/i4	/home/JohnDoe/myadded.udfm	145
DS	intra-cell	f01	/top/i3	/home/JohnDoe/intra.udfm	217
AU.BB	intra-cell	f02	/top/i3	/home/JohnDoe/intra.udfm	67

Example 9

The following examples shows how to report specific AU.TC and AU.PC faults:

```
report_statistics -detailed_analysis # you must first run this command
```

```
report_faults -class au.tc /tap_i/tap_ctrl_i/reg_enable #specified using a pin pathname
```

type	code	pin.pathname
0	AU.TC	/pad_i1/tdo_buf/tri_pin/A
0	EQ	/pad_i1/tdo_buf/ix68/Y
1	EQ	/pad_i1/tdo_buf/ix68/A
1	AU.TC	/pad_i1/tdo_buf/tri_pin/A
1	EQ	/pad_i1/tdo_buf/ix68/Y
0	EQ	/pad_i1/tdo_buf/ix68/A
0	AU.TC	/tdo
0	EQ	/pad_i1/tdo_buf/tri_pin/Y
1	AU.TC	/tdo
1	EQ	/pad_i1/tdo_buf/tri_pin/Y
0	AU.TC	/pad_i1/tdo_buf/tri_pin/E

report_faults -class au.tc 1196 #specified using a gate ID.

type	code	pin.pathname
0	AU.TC	/pad_i1/tdo_buf/tri_pin/A
0	EQ	/pad_i1/tdo_buf/ix68/Y
1	EQ	/pad_i1/tdo_buf/ix68/A
1	AU.TC	/pad_i1/tdo_buf/tri_pin/A
1	EQ	/pad_i1/tdo_buf/ix68/Y
0	EQ	/pad_i1/tdo_buf/ix68/A
0	AU.TC	/tdo
0	EQ	/pad_i1/tdo_buf/tri_pin/Y
1	AU.TC	/tdo
1	EQ	/pad_i1/tdo_buf/tri_pin/Y
0	AU.TC	/pad_i1/tdo_buf/tri_pin/E

report_faults -class au.pc test_en #specified using a pin pathname

type	code	pin.pathname
1	AU.PC	/test_en
1	AU.PC	/core_i/test_design_i/senmux/ix9/S0
0	AU.PC	/core_i/test_design_i/senmux/ix9/A0
1	AU.PC	/core_i/test_design_i/senmux/ix9/A0

report_faults -class au.pc 35 #specified using a gate ID

type	code	pin.pathname
1	AU.PC	/test_en
1	AU.PC	/core_i/test_design_i/senmux/ix9/S0
0	AU.PC	/core_i/test_design_i/senmux/ix9/A0
1	AU.PC	/core_i/test_design_i/senmux/ix9/A0

Example 10

The following example shows the format of the fault report when the fault type is specified as UDFM and you have enabled multiple detections:

report_faults

//	Class	Detections	UdfmType	Fault	Instance
//	-----	-----	-----	-----	-----
//	DS	2	intra-cell	F1	ip_inst/pio_i/i_42/inst
//	DS	1	intra-cell	F2	ip_inst/pio_i/i_42/inst
//	DS	2	intra-cell	F3	ip_inst/pio_i/i_42/inst
//	DS	3+	intra-cell	F4	ip_inst/pio_i/i_42/inst
//	DS	1	intra-cell	F5	ip_inst/pio_i/i_42/inst
//	DS	1	intra-cell	F6	ip_inst/pio_i/i_42/inst

Example 11

The following example shows the format of the fault report when you specify that you want to display faults from a maximum of two levels of hierarchy:

report_faults -hierarchy 2

// /clk_ctrl_clk_testshell	48
// /clk_ctrl_clk_testshell/gated_ccb_inst4	6
// /clk_ctrl_clk_testshell/gated_ccb_inst2	6
// /clk_ctrl_clk_testshell/i_ipoHold_10275	4
// /clk_ctrl_clk_testshell/gated_ccb_inst1	10
// /clk_ctrl_clk_testshell/antiskeew_clock_inv	4
// /clk_ctrl_clk_testshell/antiskeew_1	6
// /clk_ctrl_clk_testshell/gated_ccb_inst3	12
// /ip_inst	240366
// /ip_inst/dsp_periph_em_ccb_i	960
// /ip_inst/siox_i	8336
// /ip_inst/dio_i	25218

Related Topics

[add_faults](#)
[analyze_fault](#)
[delete_faults](#)
[read_fault_sites](#)
[read_faults](#)
[read_cpf](#)
[read_upf](#)
[report_power_data](#)
[report_testbench_simulation_options](#)
[set_fault_mode](#)
[set_fault_sampling](#)
[set_fault_subclass_analysis](#)
[set_fault_type](#)
[write_faults](#)

report_feedback_paths

Context: all contexts

Mode: all modes

Prerequisites: You can use this command only after the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode or when you issue the `create_flat_model` command.

Displays a report of currently identified feedback paths.

Usage

```
report_feedback_paths [-All | loop_id#...] [-Display {[FLAt_schematic] | [HIEarchical_schematic] | [-Hierarchical] | DAData | ALL} [APPend]] [{> | >>} file.pathname]
```

Description

Displays a report of currently identified feedback paths.

The `report_feedback_paths` command displays feedback paths the tool identified during the last circuit learning process. By default, the command displays a textual report of all currently identified feedback paths and their identification numbers. Issuing the command with the identification numbers of specific paths of interest will limit the display to just those paths.

You can use the identification numbers with the `report_loops -Display` command to schematically display specific feedback paths. When you issue this command for specific feedback paths, DFTVisualizer transcribes the same information as the `report_feedback_paths` command, but only for the specified paths.

Note

 Feedback paths include, by default, any duplicated gates.

Arguments

- **-ALL**
An optional switch that specifies to report all currently identified feedback paths. This is the default.
- ***loop_id#***
An optional, repeatable, non-negative integer that specifies the identification number of a particular feedback path to report. The tool assigns the numbers consecutively, starting with 0.
- **-Display {[FLAt_schematic] | [HIEarchical_schematic] | [Hierarchical] | DAData | ALL}**
An optional switch, repeatable literal, and literal that displays the reported information graphically in the specified DFTVisualizer window(s); if *Append* is specified, the

information is added to the existing contents of the window instead of replacing the contents.

The choices are as follows:

FLAt_schematic — A literal that specifies the Flat Schematic window.

HIErarchical_schematic — A literal that specifies the Hierarchical Schematic window.

If the Hierarchical literal is specified, only the top-level hierarchical instances are displayed in the Hierarchical Schematic window; if the Hierarchical literal is not specified, the child instances of the corresponding hierarchical instances are also displayed.

DATa — A literal that specifies the Data window.

ALl — A literal that specifies all the preceding windows.

The default when you specify -Display without including a window name is to add feedback paths to the Flat Schematic window.

- >file_pathname

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >>file_pathname

An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example enters analysis mode (which flattens the simulation model and performs the learning process), reports the identification numbers of all learned feedback paths, and then displays one of the reported feedback paths (loop #1):

```
set_system_mode analysis
report_feedback_paths

Loop#=0, feedback_buffer=26, #gates_in_network=5
    INV /I_956_I_582/ (51)
    PBUS /I_956_I_582/N1/ (96)
    ZVAL /I_956_I_582/N1/ (101)
    INV /I_956_I_582/ (106)
    TIEX /I_956_I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
    INV /I_962_I_582/ (52)
    PBUS /I_962_I_582/N1/ (95)
    ZVAL /I_962_I_582/N1/ (100)
    INV /I_962_I_582/ (105)
    TIEX /I_962_I_582/ (27)
```

```
report_feedback_paths 1 -display hierarchical_schematic
```

Related Topics

[report_loops](#)

set_loop_handling

report_flattener_rules

Context: all contexts

Mode: setup, analysis

Displays either a summary of all the flattening rule violations or the data for a specific violation.

Usage

```
report_flattener_rules [rule_id [{occurrence_id | -Verbose}]] [{> | >>} file.pathname]
```

Description

Displays either a summary of all the flattening rule violations or the data for a specific violation.

The report_flattener_rules command displays the following information for a specific violation:

- Rule identification number
- Current number of rule failures
- Violation handling

You can use the [set_flattener_rule_handling](#) command to change the handling of the net, pin, and gate rules.

Arguments

- *rule_id*

A literal that specifies the flattening rule violation for which you want to display information. The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

- Net flattening violations are described in sections “[FN1](#)” through “[FN9](#)”.
- Pin flattening violations are described in sections “[FP1](#)” through “[FP13](#)”.
- Gate flattening violations are described in sections “[FG1](#)” through “[FG8](#)”.

- *occurrence_id*

A literal that specifies the identification of the exact flattening rule violation (the occurrence) for which you want to display information. For example, you can analyze the second occurrence of the FG4 rule by specifying the rule_id and the occurrence_id, FG4 2. The tool assigns the occurrences of the rules violations as it encounters them; you cannot change either the rule identification number or the ordering of the specific violations.

- -Verbose

A switch that displays the following for each flattening rule:

- Rule identification number
- Number of failures of each rule

- Current handling status of that rule
- Brief description of that rule
- **>file_pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.
- **>>file_pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the summary information of the FG3 rule:

```
report_flattener_rules fg3
/   FG3: fails=2 handling=warning/noverbose
```

Related Topics

[set_flattener_rule_handling](#)

report_gates

Context: all contexts

Mode: setup, analysis

Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.

Usage

```
report_gates {pin_or_net_pathname | gate_id# | instance_name}...  
  {-Value_trace [position] [-TRANSition] [-Endpoints] [-DISplay]  
   [-STOP_ambiguous] | -GO_One_ambiguous | -GO_All_ambiguous]}
```

```
report_gates {-Type {gate_type | ALLF | ALL}}... [-SUMmary]
```

```
report_gates {{-Sequential_test_depth | -0Control_depth | -1Control_depth | -Observe_depth}  
  depth_number}...
```

```
report_gates {-Path {pin_pathname | gate_id} {pin_pathname | gate_id}}
```

```
report_gates {-GLitch_masked_cells}
```

```
report_gates {[ -Endpoints] [-COnstraints] [-Forward | -Backward] {pin_pathname | gate_id}...}
```

All of the above forms can redirect their output with: [{> | >>} *file_pathname*]

Description

Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.

The report_gates command displays the netlist information for either the design-level or the primitive-level gates you specify. You designate the gate by its gate identification number (ID#), a pathname of a pin connected to a gate, an instance name (design level only), or a gate type. You can specify a design cell by the pathname of a pin on the design cell. If you use a gate ID# or gate type, the command always reports the primitive-level gate. You must flatten the netlist before issuing this command.

The *pin_or_net_pathname* and *instance_name* arguments support regular expressions, which may include any number of '*' or '?' wildcard characters embedded in the pathname string. The '*' character matches any sequence of characters (including none) in a name, and the '?' character matches any single character. If a wildcard name is specified, the command will search for matching instance names from the top library cell level down to the primitive gates.

The format for the design-level gate report is:

```
instance_name      cell_type
    input_pin_name   I   (data)   pin_pathname...
...
    output_pin_name  O   (data)   pin_pathname...
...
```

The data field displays the pin information as selected by the “[set_gate_report](#)” command.

The format for the primitive-level report is:

```
instance_name      (gate_ID#)      gate_type
    input_pin_name   I   (data)   gate_ID#-pin_pathname...
...
    output_pin_name  O   (data)   gate_ID#-pin_pathname...
...
```

When you provide the name or identification number (ID#) of a user-defined ATPG function, the command reports simulation data of the function. The data reported is that specified by the Parallel_pattern option of the [set_gate_report](#) command.

The format for the function report is:

```
function_name      (function_ID#)      function_type
    input_pin_name   I   (data)   gate_ID#-pin_pathname...
...
    output_pin_name  O   (data)   gate_ID#-pin_pathname...
...
```

The list associated with the input and output pin names indicate the pins to which they connect. For the primitive level, this also includes the gate index number of the connecting gate and only includes the pin pathname if one exists at that point. There is a limitation on reporting gates at the design level. If some circuitry inside the design cell is completely isolated from other circuitry, the command only reports the circuitry associated with the pin pathname.

You can also report the fan-in or fan-out cone of a specified gate. The endpoints of a cone are defined as the primary inputs, primary outputs, tied gates, rams, roms, flip-flops, and latches. All gates reported are at the primitive level.

You can change the output of the [report_gates](#) command by using the [set_gate_report](#) command.

Note

 The tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, [report_gates](#) cannot report simulation values for the EDT logic during these cycles. This is true regardless of the option set with the [set_gate_report](#) command.

Backslashes (\), Periods (.) and Spaces in Hierarchical Names

When reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool reads the cell name, “FFH.\E/MY_SCAN_FF .SCAN_FF5” in a WGL pattern, as “FFH/E/MY_SCAN_FF/ SCAN_FF5”, and stores the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note

 The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

The following example reports on a Verilog gate named “FFH.\E/MY_SCAN_FF .SCAN_FF5”.

report_gates FFH/E/MY_SCAN_FF/SCAN_FF5

```
//  /FFH/E/MY_SCAN_FF/SCAN_FF5 FD1SF
//    D      I  /FFH/E/MY_SCAN_FF/GATE00/Z
//    A      I  /C-4/Z
//    SI     I  /FFG/SO
//    CP     I  /INPBUF1/Z
//    B      I  /C-5/Z
//    Q      O  /GATE20/C
//    QN    O
//    SO     O  /C-7/A
```

Reporting Power Data of Gates

When you issue the [read_cpf](#) or [read_upf](#) command, power data is loaded into the tool. The **report_gates** command enables power data reporting by default and shows the power on (PON) or power off (POFF) status as well as the power domain of the gate. For example:

read_upf upf_file

report_gates /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff

```
//  /lp_case_si_rst_sms/ati_rst_sync/sync_r  sync3msfqxss1ul
//    SDI    I(PON)  /vl_sms_lp_case_si_sms_proc_sms_1_stp/
U_lp_case_si_sms_proc_bist/uu5/Z
//    D      I(PON)  /lp_case_si_rst_sms/ati_rst_sync/sync_buf/Z
//    SEN    I(PON)  /se
//    CLK    I(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu1/Z
//    Q      O(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu2/A  /
lp_case_si_rst_sms/and_r/B
//    in power domain PD_P2
```

Reporting on the First Input of a Gate

The report_gates command provides a shortcut to display data on the gate connected to the first input of the previously-reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

b

The following example shows how to use report_gates and B commands to trace backward through the first input of the previously reported gate.

report_gates 26

```
// /u1/inst_565_ff_d_1_13 (26)  BUF
//   "I0"    I  269-
//   "OUT"   O  268-  75-
```

b

```
// /u1/inst_565_ff_d_1_13 (269)  LA
//   "S"    I  14-
//   "R"    I  145-
//   SCLK  I  4-/clk
//   D     I  265-/u1/_g32/X
//   ACLK  I  2-/scan_mclk
//   SDI   I  20-/u1/inst_565_ff_d_0_dff/Q2
//   "OUT"  O  26-  27-
```

b

```
// /u1/inst_565_ff_d_1_13 (14)  TIE0
//   "OUT"  O  269-  268-
```

If you include an integer argument with “b”, the trace will be backward through the specified input. For example, “b 3” will trace backward through the third input of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

For pins that are not at the library cell boundary (pins internal to the model), the pin name is enclosed in double quotes (""). The following example displays this issue.

set_gate_level primitive**report_gates /I_20/I_226/q**

```
// /I_20/I_226 dffsr
//   clk    I  (HX)  /I_20/I_225/out
//   d      I  (X)   /I_20/I_222/out
//   pre   I  (H1)  /PRE
//   clr   I  (H1)  /CLR
//   q     O  (X)   /I_16/io  /I_23/I_221/io  /I_6/io
//   qb   O  (X)
```

set_gate_level primitive

```
// Creating schematic for 5 instances (1 was compacted).
```

report_gates /I_20/I_226/q

```
// /I_20/I_226 (12) BUF
//   "I0" I (0) 39-
//   q     O (X) 16-/I_16/io 31-/I_23/I_221/io
//           17- /I_6/io
```

b

```
// /I_20/I_226 (39) DFF
//   "S" I (LX) 26-
//   "R" I (LX) 23-
//   clk I (HX) 20-/I_20/I_225/out
//   d   I (X) 36-/I_20/I_222/out
//   "OUT" O (0) 12- 13-
//   MASTER cell_id=1 chain=c1 group=g1 invert_data=FFFF
```

Reporting on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also use a shortcut to trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

f

The following example shows how to use Report Gate and F commands to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
// /u1/inst_565_ff_d_1_13 (269) LA
//   "S" I 14-
//   "R" I 145-
//   SCLK I 4-/clk
//   D   I 265-/u1/_g32/X
//   ACLK I 2-/scan_mclk
//   SDI I 20-/u1/inst_565_ff_d_0_dff/Q2
//   "OUT" O 26- 27-
```

f

```
// /u1/inst_565_ff_d_1_13 (26) BUF
//   "I0" I 269-
//   "OUT" O 268- 75-
```

f

```
// /u1/inst_565_ff_d_1_13 (268) LA
//   "S" I 14-
//   "R" I 145-
//   BCLK I 1-/scan_sclk
//   "D0" I 26-
//   "OUT" O 24- 25-
```

If you include an integer argument with “f”, the trace will be forward through the specified fanout of the previously reported gate. For example, “f 2” will trace forward through the second fanout of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

Note

 When reporting gates, be aware that a reported X may not have been captured. For example, when the tool uses scan chain masking, it records the actual measured value for each cell only in the unmasked, selected scan chain in a compactor group; the rest of the scan chains in the group are masked, so the tool records all Xs for their scan cells. When not using scan chain masking, the tool records an X for a scan cell if it is made unmeasurable as a result of the actual occurrence of an X in the corresponding cell of a different scan chain in the same compactor group. For more information, see “[Understanding Scan Chain Masking in the Compactor](#)” in the Tesson TestKompress User’s Manual.

When using the tool to report on RAM or ROM gates, the report_gates command displays the RAM and ROM data that describes their behavior. You can use the set_gate_report command to customize the content of the reported data. For example, assume you add_faults on a RAM pin, then generate RAM sequential patterns. If you follow that with a set_gate_report command, specifying the Parallel_pattern argument, the report_gates output shows for the specified pattern, the address and data written to the RAM. A hyphen (-) separates the information for each cycle. The following example shows how this information is displayed for an 8-bit output named “OUT” and a 4-cycle pattern.

```
"OUT" O (S [0]=00000011-S [0]=00001001-S-SX) 5353- 5354- 5355-
      5356- 5357- 5358-
      5359- 5360-
```

In the first cycle, the pattern writes 00000011 to address 0, in the second, it writes 00001001 to address 0. The pattern does not write anything in the third cycle. The fourth cycle is a “don’t care” cycle.

The RAM and ROM simulation primitives are the same as the library primitives with the outputs being OUT gates in the RAM/ROM fanout list. The report_gates command gives RAM behavior summary information at the end of the displayed data. The report displays messages similar to the following:

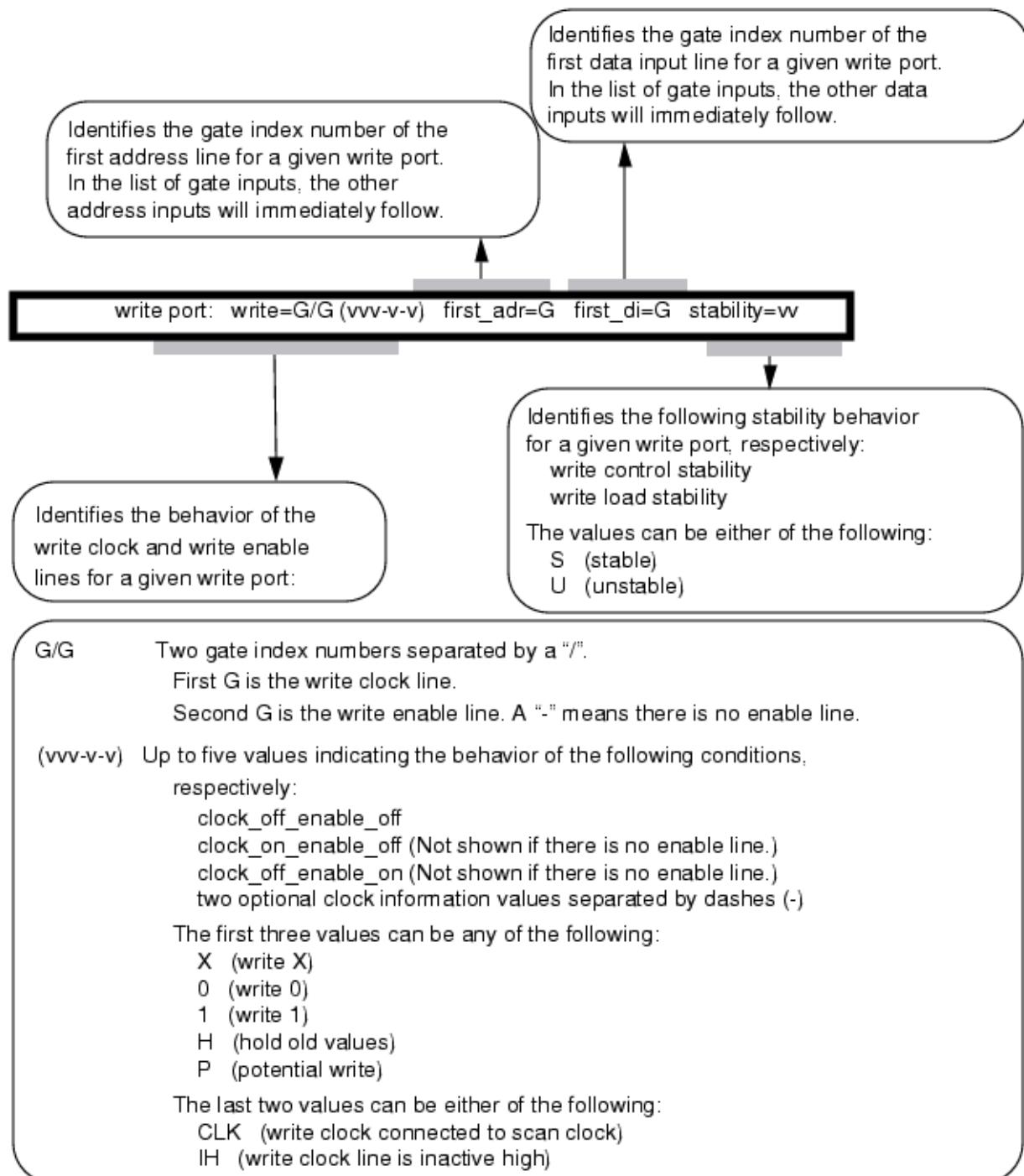
```
write port: write=G/G (vvv-v-v) first_addr=G first_di=G stability=vv
read port: read=G/G (vvv-v-v) first_addr=G first_do=G stability=vv
Test behavior: stability=vvvv tiex-X=v read_only=v
                  ram_sequential=v/v seq_transparent=v/v
Contention Behavior: write_write=v/v read_read=v read_write=v/v
```

A RAM is classified as a read-only RAM if all three of the following conditions are met:

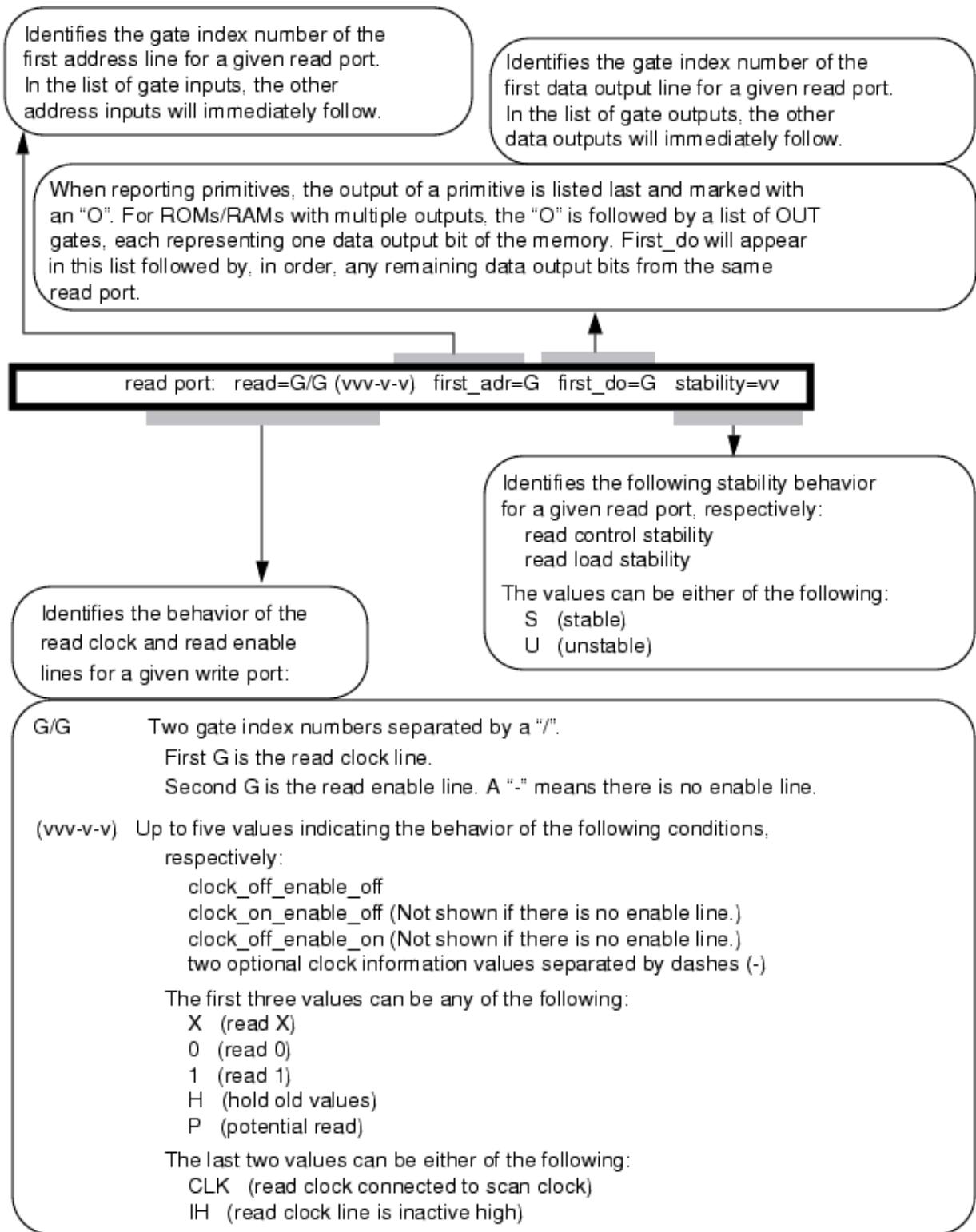
- its contents cannot be disturbed by any procedure except test setup;
- its set, reset and write ports must hold at an inactive state in the capture window;

- it has an initialization file defined in the Tessent Cell library.

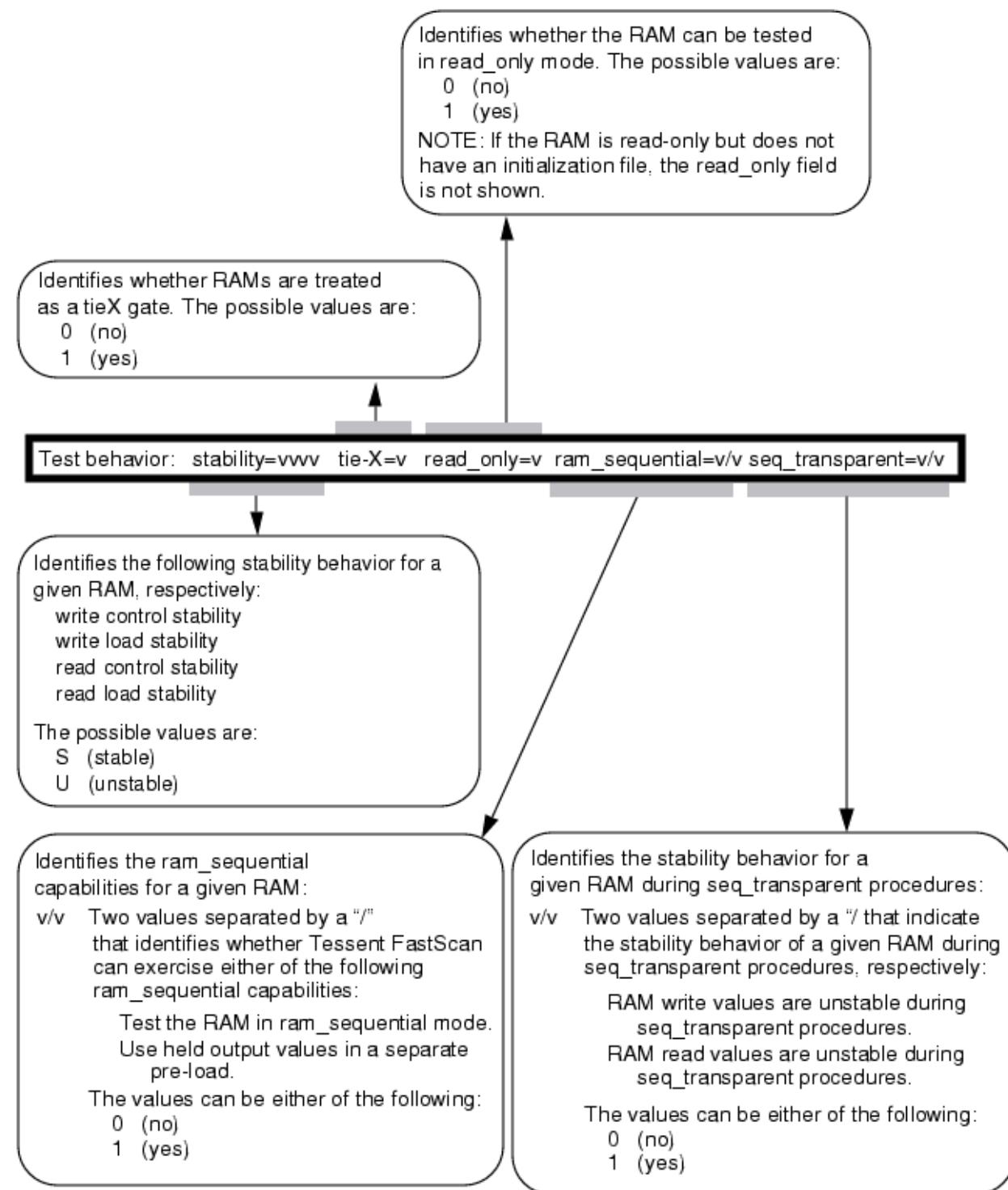
The following describes the fields for the write port message line:



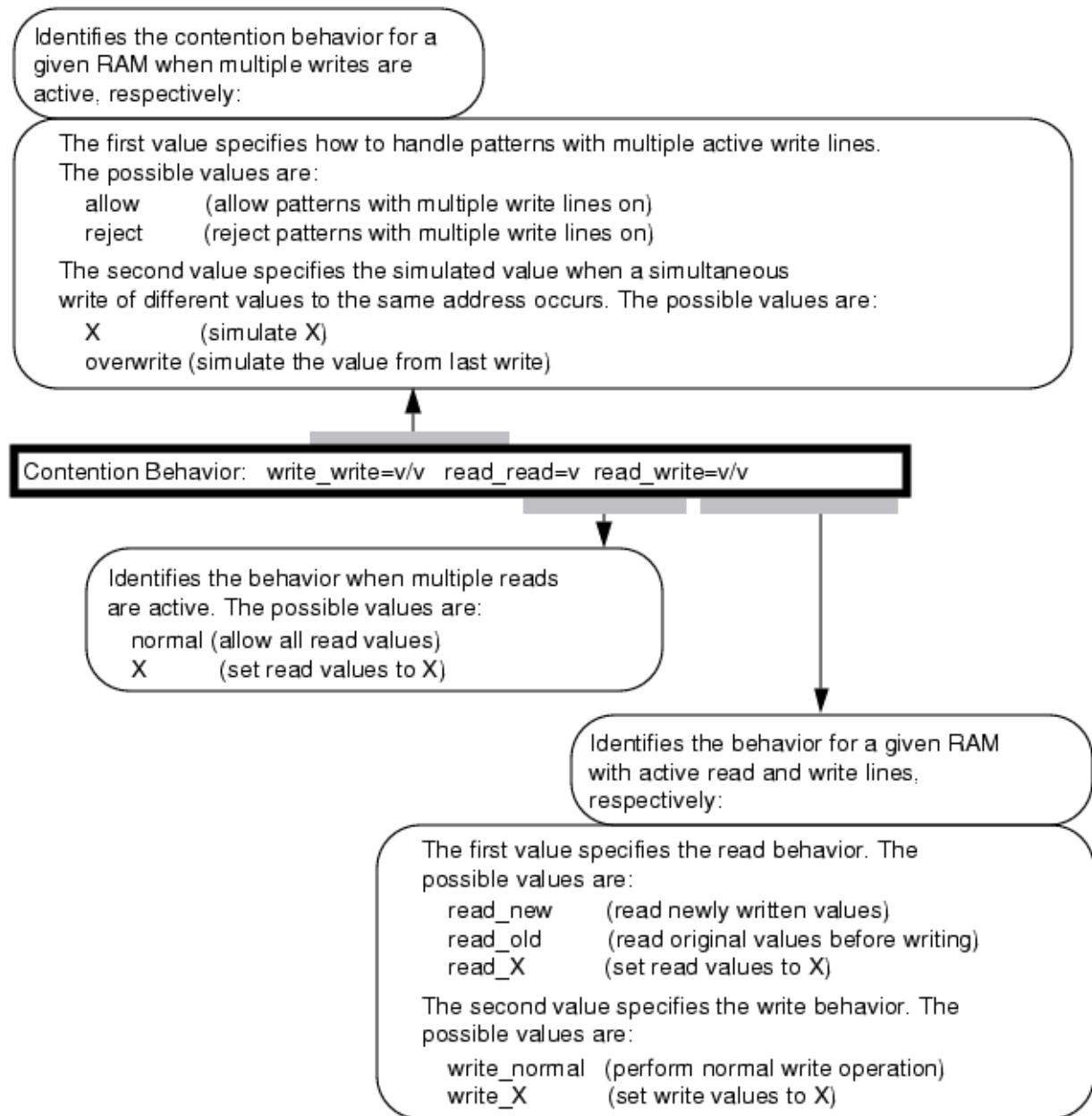
The following describes the fields for the read port message line:



The following describes the fields for the Test behavior message line:



The following describes the fields for the Contention Behavior message line:



Arguments

The following four arguments list the four methods for naming the objects on which you want the tool to display information. You can use any number of the four argument choices, in any order.

- ***function_name***

A repeatable string that specifies a user-defined ATPG function.

- **gate_id#**

A repeatable integer that specifies the gate identification numbers of the objects. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

The tool also accepts gate_pin IDs specified in the following format:

'*gate_id:port_index*'

- **pin_or_net.pathname**

A repeatable string that specifies the pathnames of pins or nets in the design netlist. You may use wildcard characters to match multiple pin or net pathnames.

For a hierarchical pathname, the display will include information describing how that pathname maps to the driving design level pin(s) and gate(s) for which data is displayed.

- **instance_name**

A repeatable string that specifies the hierarchical pathname of an instance of a library cell within the design. If a valid library instance pathname is given when in primitive level, all pins on that library cell are reported. When in primitive level, *instance_name* may also be the pathname of a primitive instance.

- **-Value_trace [position] [-TRANsition] [-Endpoints] [-DISplay] [-STOP_ambiguous | -GO_One_ambiguous | -GO_All_ambiguous]**

An optional switch that specifies to trace backward from the output pin of the gate to the source of a value. The -Value_trace option only supports the primitive report level; therefore, all arguments must be specified at the primitive level.

position

An optional value that specifies which bit to trace on for multi-bit data. For example, if a clock signal is 0X0 for a shift cycle, you can specify the tracing is on the second bit (X). The default is to use position 0 (the first bit).

-TRANsition

An optional switch that specifies to trace backward to the source of the transition of a pair of values. When using the *position* option, *position* specifies the first bit of the transition.

-Endpoints

An optional switch that specifies to report only endpoints that control the specified gate. The result of using this option also depends on the tracing mode. Only when using the -GO_All_ambiguous tracing mode are all endpoints reported.

-DISplay

An optional switch that displays all tracing gates in the Flat Schematic window of DFTVisualizer.

Note

The -Display switch cannot be used in conjunction with the -Path switch.

The following three optional switches let you choose the tracing mode of operation.

-STOP_ambiguous

An optional switch that specifies to stop if either input could cause the value on the output. This is the default setting if -go_one_ambiguous or -go_all_ambiguous is not specified.

-GO_One_ambiguous

An optional switch that specifies to trace one input if multiple input could cause the output value. For example, if the output of an OR is X, input A is X, input B is X, then the source of X is represented by both A and B. The tool will pick one controlling input, say B, and continue tracing input B.

then the source of X is represented by B. The tool will continue tracing input B.

-GO_All_ambiguous

An optional switch that specifies to trace all inputs if multiple input could cause the output value.

- **-Type {gate_type | ALLF | ALL}**

An optional repeatable switch where *gate_type* is a name pair that specifies the gate types for which you want to display the gate information. [Table 5-12](#) lists the supported types.

The **ALLF** option lets you report gates on all primitive level gates and ATPG functions. This feature supports users who require access to the tool's flat models. The **ALL** option lets you report gates on all primitive level gates (but not functions).

- **-SUMmary**

An optional switch that reports only the number of gates found when using the -Type switch.

- **-Forward {pin_pathname | gate_id}...**

An optional switch that reports the fan-out cone of the specified gate.

- **-Backward {pin_pathname | gate_id}...**

An optional switch that reports the fan-in cone of the specified gate.

- **-Endpoints [-Forward | -Backward] {pin_pathname | gate_id}...**

An optional switch that reports only the endpoint of the cone.

Note

 Immediately after -Endpoints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

- -COnstraints [-Forward | -Backward] {pin.pathname | gate_id}...

An optional switch that takes into account the effects of pin and cell constraints.

For a gate whose output is constrained to a fixed value, the tools report only other gates whose output is also constrained. For a gate whose output is not constrained to a fixed value, the tools report only other gates whose outputs are not constrained. During backwards tracing, a mux input, which is always blocked by a constrained value on the select line of the mux, will never be backtraced.

Note

 Immediately after -Constraints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

- -Path {gate_id# | pin.pathname} {gate_id# | pin.pathname}

Note

 This switch cannot be used in conjunction with the -Display switch.

An optional switch that reports on the path between the first gate_id# or pin.pathname and the second gate_id# or pin.pathname. All paths must be specified at the primitive level. Paths do not extend through sequential elements. The output generates a report on each primitive in the path(s), in order of increasing gate_id#.

- -GLitch_masked_cells

An optional switch that specifies for the tool to report the sequential elements that have been masked due to glitches on the set or reset port. It also reports the number of patterns for which the cells are masked. This is an example of the report:

```
// 30 state elements are masked due to set/reset port glitches,  
// which may impact the test coverage.  
// #-pat-cycles-masked state-element  
// -----  
// 1 '/cm0507/cm2304/' (7956)  
// 1 '/cm0507/cm2305/' (7959)
```

- -Sequential_test_depth *depth_number*

An optional switch and integer pair that specifies for the tool to report gates with a sequential test depth greater than the specified *depth_number*. The sequential test depth is the greater of the control 0 and control 1 depths plus the observe depth for the primitive

output. This is the minimum depth required to test stuck faults at the primitive output. This is also the number used to determine the maximum sequential test depth of the circuit.

- **-0Control_depth *depth_number***

An optional switch and integer pair that specifies for the tool to report gates with a control 0 depth greater than the specified *depth_number*. The control 0 depth is the minimum sequential depth required to set the primitive output to a 0.

- **-1Control_depth *depth_number***

An optional switch and integer pair that specifies for the tool to report gates with a control 1 depth greater than the specified *depth_number*. The control 1 depth is the minimum sequential depth required to set the primitive output to a 1.

- **-Observe_depth *depth_number***

An optional switch and integer pair that specifies for the tool to report gates with an observe depth greater than the specified *depth_number*. The observe depth is the minimum sequential depth required to observe the output of the primitive at either a primary output or scan cell.

- **> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>> *file_pathname***

An optional redirection operator and pathname pair used, at the end of the argument list, for appending to the contents of *file_pathname*.

Reference Tables

- These are the reference tables for gate types and clock port categories:
 - [Table 5-12](#) lists all of the types of gates that you can specify with the -Type switch.
 - [Table 5-13](#) lists the various categories of clock ports on state elements that can appear in report_gates output.

Table 5-12. Reportable Gate Types

gate_type	Description
BUF	buffer
INV	inverter
AND	and
NAND	inverted and
OR	or
NOR	inverted or
XOR	exclusive-or

Table 5-12. Reportable Gate Types (cont.)

gate_type	Description
NXOR	inverted exclusive-or
DFF	D flip-flop, same as _dff library primitive
LA	latch, same as _dlat library primitive
PI	primary input
PO	primary output
TIE0	tied low
TIE1	tied high
TIEX	tied unknown
TIEZ	tied high impedance
HIST	histogram of each primitive type used
TSD	tri-state driver, first input is active high enable line
BUS	tri-state bus
ZVAL	Z converter gate, converts Z to X
WIRE	undetermined wired gate
MUX	2-way multiplexer, first line is select line
SW	switch gate, first input is active high enable line
PBUS	pulled bus gate, where the second input is the pulled value
OUT	memory model gate, created for each read data bit
RAM	random access memory
ROM	read only memory
XDET	X detector, gives 1 when input is X
ZDET	Z detector, gives 1 when input is Z
TLA	transparent latch
STLA	seq_transparent latch
STFF	seq_transparent flip-flip
FB_BUF	internal gate type to break combinational loops in the design
MASTER	the primary memory element of a scan cell
SLAVE	an independently-clocked scan cell memory element; resides in the scan chain path.

Table 5-12. Reportable Gate Types (cont.)

gate_type	Description
SHADOW	non-scan flip-flop that has the D input connected to the Q output of a scan flip-flop
COPY	a memory element that lies in the scan chain path and can contain the same (or inverted) data as the associated master or slave element in the scan cell

Table 5-13. Clock Port Categories

Category	Description
IL	inactive low
IH	inactive high
AHS	active high slave
ALS	active low slave

For more information about using the clock port options, refer to “[The ATPG Analysis Option](#)” on page 2692.

Examples

Example 1

The following example displays the simulated values of the gate and its inputs at the primitive level:

```
set_system_mode analysis
set_gate_report error_pattern
set_gate_level primitive
report_gates i_1006/o
```

The gate report for the design level may look like the following:

```
/P2.13P ND2
A I /LD.1
B I /M1.1
Z O /P2.2P/S
```

The gate report for the primitive level may look like the following:

```
/P2.13P (23) NAND
A I 9-/LD.1
B I 6-/M1.1
Z O 33-/P2.2P/S
```

The gate report for the primitive level of a RAM gate may look like the following:

```
//  /P1.RAM/U1  (67)  RAM
//    "I0"    I  27-
//    "I1"    I  20-
//    RCK0   I  36-
//    "I3"    I  26-
//    "I4"    I  42-
//    "I5"    I  43-
//    "I6"    I  44-
//    "I7"    I  45-
//    "I8"    I  46-
//    WCK0   I  28-
//    "I10"   I  19-
//    A14    I  29-
//    A13    I  30-
//    A12    I  31-
//    A11    I  32-
//    A10    I  34-
//    D14    I  66-/P1.RAM/D1[0]
//    D13    I  65-/P1.RAM/D1[1]
//    D12    I  64-/P1.RAM/D1[2]
//    D11    I  63-/P1.RAM/D1[3]
//    D10    I  62-/P1.RAM/D1[4]
//    "OUT"   O  68-  69-  70-
//                  71-  72-
//    address size =      5
//    data size =       5
//    minimum address =    0
//    maximum address =   31
//    number write ports =  1
//    number read ports =  1
//    edge_triggered =     no
//    initialization file = ram.init_file
//    write port: write=28/- (H)  first_adr=29  first_di=66
//                                stability=SS
//    read port:  read=36/- (0)  first_adr=42  first_do=68
//                                stability=SS
//    Test behavior: stability=SSSS  tie-X=0  read_only=1
//                                ram_sequential=1/0  seq_transparent=0/0
//    Contention behavior: write_write=allow/X  read_read=normal
//                                read_write=read_new/write_normal
```

Example 2

The following example defines a function named “my_function” and displays all its simulation results from the last simulation pass:

```
set_system_mode analysis
add_atpg_functions my_function or /g1 ~/g2
set_gate_report parallel_pattern all
report_gates my_function

//  my_function  (44)  USER_OR
//    "I0"    (0/0/0/0/0/0/0/0.../0) 12-/g1
//    ~"I1"    (1/1/1/1/1/1/1.../1) 13-/g2
//    "OUT"   (0/0/0/0/0/0/0.../0)
```

Example 3

The following example shows the use of wildcards:

```
report_gates /xscan_0_0_cch_scan_32x/ix15*
// /xscan_0_0_cch_scan_32x/ix151 NOR2XL
// A I /myop1<0> [1]
// B I /myop2[1]
// Y O /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A0
// /xscan_0_0_cch_scan_32x/ix153 NAND2X1
// A I /myop1<0> [0]
// B I /myop2[0]
// Y O /xscan_0_0_cch_scan_32x/sum_add_0_ix1/A \
// /xscan_0_0_cch_scan_32x/ sum_add_0_ix23/B \
// /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A1
// /xscan_0_0_cch_scan_32x/ix155 NAND2X1
// A I /myop1<0> [1]
// B I /myop2[1]
// Y O /xscan_0_0_cch_scan_32x/ix166/B0 \
// /xscan_0_0_cch_scan_32x/ sum_add_0_ix27/B0
// /xscan_0_0_cch_scan_32x/ix157 NAND2X1
// A I /myop1<0> [3]
// B I /myop2[3]
// Y O /xscan_0_0_cch_scan_32x/ix174/B0 \
// /xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```

```
report_gates /xscan_0_0_cch_scan_32x/ix159
```

```
// /xscan_0_0_cch_scan_32x/ix159 NAND2X1
// A I /myop1<0> [5]
// B I /myop2[5]
// Y O /xscan_0_0_cch_scan_32x/ix188/B0 \
// /xscan_0_0_cch_scan_32x/ sum_add_0_ix83/B0
```

set_gate_level primitive

```
report_gates /xscan_0_0_cch_scan_32x/ix157
```

```
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A I 7-/myop1<0> [3]
// B I 17-/myop2[3]
// Y O 60-/xscan_0_0_cch_scan_32x/ix174/B0 \
// 75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A I 7-/myop1<0> [3]
// B I 17-/myop2[3]
// Y O 60-/xscan_0_0_cch_scan_32x/ix174/B0 \
// 75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A I 7-/myop1<0> [3]
// B I 17-/myop2[3]
// Y O 60-/xscan_0_0_cch_scan_32x/ix174/B0 \
// 75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```

Example 4

The following example shows how the output report will change if the input pathname is a hierarchical pin or net. In this case, an additional line is output at the top of the report, indicating the mapping that was found.

set_gate_level design

report_gates /sub3/in2

```
// Hierarchical pin /sub3/in2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//   A1 I /in1
//   A0 I /sub1/gate2/Y
//   Y O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//     /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

report_gates /w2

```
// Hierarchical net /w2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//   A1 I /in1
//   A0 I /sub1/gate2/Y
//   Y O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//     /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

Example 5

The following example traces backward to the source of a DRC error:

```
// Error: clock input of /ff2 (35) set to X. (T5-1)
// Error: Scan chain c0 blocked at gate /ff2 (35) after tracing 1 cells.
// (T3-1)
// Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

set_gate_level prim

set_gate_report drc shift

report_gates 35

```
// /ff2 (35) DFF
//   "S" I (000) 10-
//   "R" I (000) 9-
//   CLK I (0X0) 27-/gate8/Y
//   "D0" I (XXX) 32-
//   "OUT" O (XXX) 14- 15-
```

report_gates 27 -v 1

```
//  /gate8 (27)  AND
//    A0      I  (010)  6-/clk1
//    A1      I  (XXX)  21-/ff4/Q
//    Y       O  (0X0)  35-/ff2/CLK
//
//  /ff4 (21)  INV
//    "I0"    I  (XXX)  11-
//    Q       O  (XXX)  27-/gate8/A1
//
//  /ff4 (11)  INV
//    "I0"    I  (XXX)  33-
//    "OUT"   O  (XXX)  21-  22-
//
//  /ff4 (33)  DFF
//    "S"     I  (000)  10-
//    "R"     I  (000)  9-
//    CLK     I  (010)  7-/clk2
//    D       I  (XXX)  3-/in3
//    "OUT"   O  (XXX)  11-
//
// Total 4 gates traced which controls gate /gate8 (27) to X (bit 1).
```

Example 6

The following example traces backward to the first ambiguous gate where multiple source control the values:

```
set_gate_level prim
set_gate_report drc shift
report_gates 29

//  /sgate1 (29)  AND
//    A0      I  (XXX)  25-/sgate2/Y
//    A1      I  (111)  10-
//    Y       O  (XXX)  31-/ff1/SI

report_gates29 -v 1

//  /sgate1 (29)  AND
//    A0      I  (XXX)  25-/sgate2/Y
//    A1      I  (111)  10-
//    Y       O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//    A0      I  (XXX)  21-/ff0/Q
//    A1      I  (XXX)  16-/ff2/QB
//    Y       O  (XXX)  29-/sgate1/A0
// Total 2 gates traced which controls gate /sgate1 (29) to X (bit 1).
```

Example 7

The following example traces backward to all ambiguous gates:

```
report_gates 29 -v 1 -go_all
```

```

//  /sgate1 (29)  AND
//    A0      I  (XXX)  25-/sgate2/Y
//    A1      I  (111)  10-
//    Y       O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//    A0      I  (XXX)  21-/ff0/Q
//    A1      I  (XXX)  16-/ff2/QB
//    Y       O  (XXX)  29-/sgate1/A0
//
//  /ff0 (21)  INV
//    "I0"    I  (XXX)  12-
//    Q       O  (XXX)  25-/sgate2/A0
//
//  /ff0 (12)  INV
//    "I0"    I  (XXX)  35-
//    "OUT"   O  (XXX)  21-  22-
//
//  /ff0 (35)  DFF
//    "S"     I  (000)  11-
//    "R"     I  (000)  9-
//    CLK     I  (010)  6-/clk1
//    D       I  (XXX)  3-/in3
//    "OUT"   O  (XXX)  12-
//
//  /ff2 (16)  INV
//    "I0"    I  (XXX)  37-
//    QB     O  (XXX)  25-/sgate2/A1
//
//  /ff2 (37)  DFF
//    "S"     I  (000)  11-
//    "R"     I  (000)  9-
//    CLK     I  (010)  6-/clk1
//    "D0"    I  (XXX)  34-
//    "OUT"   O  (XXX)  15-  16-
// Total 7 gates traced which controls gate /sgate1 (29) to X (bit 1).

```

Example 8

The following example traces backward from the output of gate 122 to the origin of the transition:

```

// Fault type is transition

set_gate_level prim
set_gate_report pattern_index 3
report_gates 122 -value_trace -transition

```

```
// /top_i/sf_0002_2 (122) MUX
//   SE    I  (111-111)  6-/sen
//   D     I  (111-000)  3-/pi[0]
//   SI    I  (011-111)  36-/top_i/sf_0001_2/Q
//   "OUT"  O  (011-111)  419-
// /top_i/sf_0001_2 (36)  BUF
//   "I0"   I  (011-111)  418-
//   Q     O  (011-111)  115-/top_i/an_2/A2  122-/top_i/sf_0002_2/SI
// /top_i/sf_0001_2 (418)  DFF
//   "S"    I  (000-000)  12-
//   "R"    I  (000-000)  136-
//   CLK   I  (010-000)  5-/clk
//   "D0"   I  (111-111)  404-
//   "OUT"  O  (011-111-[1]) 36- 37-
//   MASTER cell_id=5 chain=chain1 group=grp1 invert_data=FFFF
//
// Total 3 gates traced which control gate '/top_i/sf_0002_2/' (122) to
L_H transition (bit 0-1).
```

Example 9

The following example displays the status of the faults on the gate and its inputs:

```
set_gate_report fault_status
report_gates /ram1/address<4>

// /ram1 picdram
//   clk    I  (AU.SEQ:AU.SEQ)  /gater2/an_4/Y
//   address<6> I  (AU.SEQ:AU.SEQ)  /address1[6]
//   address<5> I  (AU.SEQ:AU.SEQ)  /address1[5]
//   address<4> I  (AU.SEQ:AU.SEQ)  /address1[4]
//   address<3> I  (AU.SEQ:AU.SEQ)  /address1[3]
//   address<2> I  (AU.SEQ:AU.SEQ)  /address1[2]

report_gates /an_10/A1

// /an_10 and03
//   A0    I  (AU.BB:AU.BB)  /sf_1_6/Q
//   A1    I  (AU.BB:AU.BB)  /sf_3_6/Q
//   A2    I  (AU.BB:AU.BB)  /sf_5_6/Q
//   Y     O  (AU.BB:AU.BB)  /po6

report_gates /gater11/an_1/Y

// /gater11/an_1 inv02
//   A    I  (AU.TC:AU.TC)  /an_13/Y
//   Y    O  (AU.TC:AU.TC)  /gater11/lat1/CLK
```

In the example, the tool reports the following untestable (AU) fault sub-classes:

- BLACK_BOXES (BB) — Untestable faults due to black boxes.
- SEQUENTIAL_DEPTH (SEQ) — Untestable faults with the current sequential_depth and may be testable with a higher depth.
- TIED_CELLS (TC) — Untestable faults due to tied or blocked tied non-scan cells.

For a complete list of these AU fault sub-classes, see [Table 6-18](#).

Example 10

The following example shows how a launch-off-shift transition pattern can be created and reported:

```
set_fault_type transition -allow_shift_launch
create_patterns
set_gate_report pattern_index 0
report_gates /top_i/sf_0001_1

// /top_i/sf_0001_1 sffr
//   D      I  ((0)-111)  /pi[0]
//   SI     I  ((1)-000)  /top_edt_i/top_edt_bypass_logic_i/U18/Y
//   SE     I  ((1)-000)  /sen
//   CLK    I  ((0)-010)  /clk
//   R      I  ((1)-111)  /rst
//   Q      O  ((1)-111-[1])  /top_i/an_2/A1  /top_i/sf_0002_1/SI
//   QB    O  ((0)-000-[0])
```

In the example, the first value in parentheses is the value from the last scan-shift event. The next values (following the dash “-”) are the first capture-cycle values. In this case there are three events for the first capture cycle.

Example 11

The following example shows how the value used to simulate a gate is displayed (in **bold**) when that gate is reported as not simulated:

```
// command: rep gate /piccpu_i/status_reg_7/DFF1/D      // not simulated
// /piccpu_i/status_reg_7/DFF1 (2520)  DFF
//   "S"    I  66-
//   "R"    I  67-
//   CP    I  543-
//   D      I  2301-/piccpu_i/status_reg_7/MUX2/Q
//   "OUT"  O  235-
//   MASTER  cell_id=0  chain=chain7  group=grp1  invert_data=FFFF
//
// Proc: shi 1 sh 2 sh 3 sh 4 sh 5 sh 6 sh 7 sh 8 sh 9 sh10 sh11 cap
// -----
//   i
//   n 123  123  123  123  123  123  123  123  123  123  123  o  o
// Time: i 000  000  000  000  000  000  000  000  000  000  000  f  f
//       t0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  fxf
// -----
// Sim: XXXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XX11 1XX
//
// Inputs:
// "S"    00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 000
// "R"    00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 000
// CP    00010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0X0
// D    was not simulated during shift procedure in DRC K19. Simulation
// value used = X.
```

Example 12

The following example shows how to report only the total number of master gates:

```
ANALYSIS> report_gates -type master -summary  
// Total number of master gates = 376034
```

Example 13

This example demonstrates how to report the simulated values from all natural, tied gates, and learned constant value non-scan cells.

Use the “set_gate_report -constrain_value on” command. Then report_gates is used.

The report shows that the INV gate in /sdff2/lat is unreachable from any observation point, as indicated by UU value.

```
set_gate_report -constrain_value on  
report_gates sdff2/lat  
  
// /sdff2/lat (18)  BUF  
//   "I0"    I  (X/01/-)  34-  
//   Q      O  (X/01/-)  36-/so  
// /sdff2/lat (19)  INV  
//   "I0"    I  (X/01/-)  34-  
//   QN    O  (X/01/-/UU)  
// /sdff2/lat (34)  TLA  
//   "S"    I  (0/1/-)  9-  
//   "R"    I  (0/1/-)  8-  
//   CLK    I  (-/-/B)  26-/sdff2/u0/OUT  
//   D      I  (-/-/-)  29-/sdff2/u1/OUT  
//   "OUT"  O  (X/01/-)  18-  19-  
// Reported all gates in instance 'sdff2/lat'
```

Example 14

This procedure file defines four events in the shift procedure (force_pi is set to occur at 5 ns rather than at the beginning of the cycle).

report_gates

```

set time scale 1.000000 ns ;
set strobe_window time 1 ;
timeplate gen_tp1 =
    force_pi 5 ;
    measure_po 10 ;
    pulse CLOCK 20 10;
    period 40 ;
end;
procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        measure_sco ;
        pulse CLOCK ;
    end;
end;
procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force scan_en 1 ;
        force CLOCK 0 ;
    end ;
    apply shift 3;
end;
procedure test_setup =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force IN1 0 ;
    end;
end;

```

The report_gates command reports four values, one for each event in the cycle.

```

set_gate_level design
set_gate_report drc shift
report_gates .reg1

// /reg1 dfsc
// CK I (0010) /CLOCK
// D0 I (XXXX) /gate1/OUT
// D1 I (SSSS) /reg2/QBAR
// SC I (1111) /scan_en
// Q O (XXXX) /gate2/B
// QBAR O (SSSS) /scan_out1

```

Related Topics

[set_gate_level](#)

[set_gate_report](#)

report_graybox_statistics

Context: patterns -scan

Mode: analysis

Reports the statistics gathered by graybox analysis.

Usage

```
report_graybox_statistics {[ -top integer] | [-percentage_exceeds integer] |  
[-exceeds_max_sequential_levels]}
```

Description

Reports the statistics gathered by graybox analysis. This command works only after running graybox analysis using the -collect_reporting_data switch of the [analyze_graybox](#) command.

Arguments

- **-top *integer***

An optional switch and integer pair that lists the graybox analysis backward tracing origins with the highest number of leaf instances. The *integer* argument specifies how many tracing origins from the top will be listed in the report output. A tracing origin can be a PO, a wrapper cell input pin, or a preserve instance.

- **-percentage_exceeds *integer***

An optional switch that lists all tracing origins that exceed the specified percentage of leaf instances (with respect to all leaf instances in design) covered during backtracing.

- **-exceeds_max_sequential_levels**

An optional switch that lists the tracing origins that exceed the maximum sequential levels limit specified by “analyze_graybox -max_sequential_levels”. The default value for maximum sequential levels is 2 unless specified by the -max_sequential_levels switch.

Examples

The following example shows the command output for the top four tracing origins with the highest number of leaf instances:

```
report_graybox_statistics -top 4
```

Start Point	Pin Type	Number of Leaf Instances	Percentage of Design	Sequential Levels Limit Reached
sff2/D, SI	WrapperCellDataPin	3	50.00	yes
sff1/D	WrapperCellDataPin	2	33.33	-
po	PrimaryOutput	1	16.67	-
so	PrimaryOutput	1	16.67	-

The explanation of the columns are as follows:

- Start Point is the pathname of the tracing origin.
- Pin Type is one of the following: PrimaryOutput, WrapperCellDataPin, WrapperCellSetPin, WrapperCellResetPin, WrapperCellClockPin, PreserveInstanceInputPin.
- Number of Leaf Instances is the number of leaf instances marked for the tracing origin.
- Percentage of Design is the percentage of covered leaf instances with respect to the leaf instances in the entire design.
- Sequential Levels Limit Reached is “yes” when the specified or default maximum sequential levels limit is exceeded, and is “-” otherwise.

Related Topics

[analyze_graybox](#)

report_icl_extraction_options

Context: all contexts

Mode: all modes

Provides a human readable report of the ICL extraction options.

Usage

```
report_icl_extraction_options
```

Description

Provides a human readable report of the ICL extraction options.

Arguments

None

Examples

The following command displays the ICL extraction settings in a human readable form:

```
report_icl_extraction_options
//  Option          | Type | Value
//  -----+-----+-----+
//  extract_learned_muxes | BOOL |    on
//  extract_library_muxes | BOOL |    on
```

Related Topics

[get_icl_extraction_options](#)

[set_icl_extraction_options](#)

report_icl_modules

Context: dft, patterns

Mode: all modes

Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.

Usage

```
report_icl_modules [-all | -modules modules [-hierarchical]]
```

Description

Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.

If you invoke this command without any arguments, the names of all loaded and extracted ICL modules are reported in human readable form.

Note

 If you invoke this command during setup mode, before ICL extraction has taken place, only the loaded ICL modules will be reported. If you invoke this command in the analysis mode, the tool may have performed ICL extraction, and the report_icl_modules command will report the loaded and extracted ICL modules.

Arguments

- **-all**
An optional switch that reports all loaded and extracted ICL modules in ICL syntax.
- **-modules *modules***
An optional switch that reports the specified modules in ICL syntax. The switch can take an *object_spec* (TCL string list or object list).
- **-hierarchical**
An optional switch that reports the listed modules, as well as a recursive listing of all modules that are instantiated by the listed modules and their child modules.

Examples

Example 1

In the following example, when you issue this command without any arguments, it reports the names of all the ICL modules in human readable form:

```
report_icl_modules
```

```
//      pll1
//      pll2
//      pll3
//      pll4
//      block
```

If the current design is set when you issue this command, the current design will be marked as shown in the following example:

```
set_current_design top
report_icl_modules

//      pll1
//      pll2
//      pll3
//      pll4
//      block
//      top      (current iProcsForModule ICL module)
//      Middle
//      bottom
```

Example 2

The following example reports the ICL syntax for the module chip:

```
report_icl_modules -modules chip
```

```
Module chip {
    // ICL module read from source on or near line 1 of file './data/icl/
    chip3.icl'
    TCKPort tck;
    ScanInPort tdi;
    ScanOutPort tdo {
        Source MyTap.tdo;
    }
    TMSPort tms;
    TRSTPort trst;
    Instance MyTap Of tap {
        InputPort tck = tck;
        InputPort tdi = tdi;
        InputPort tms = tms;
        InputPort trst = trst;
        InputPort fromTdr1 = MySib2.so;
    }
}
```

Example 3

The following example reports the ICL syntax for the module chip as well as a recursive listing of all modules instantiated by chip and its child modules.

```
report_icl_modules -modules chip -hierarchical
```

```
Module chip {
    // ICL module read from source on or near line 1 of file './data/icl/
chip.icl'
    TCKPort tck;
    ScanInPort tdi;
    ScanOutPort tdo {
        Source MyTap.tdo;
    }
    TMSPort tms;
    TRSTPort trst;
    Instance MyTap Of tap {
        InputPort tck = tck;
        InputPort tdi = tdi;
        InputPort tms = tms;
        InputPort trst = trst;
        InputPort fromTdr1 = MySib2.so;
    }
    Instance MySib1 Of sib {
        InputPort si = tdi;
        InputPort se = MyTap.se;
        InputPort ce = MyTap.ce;
        InputPort ue = MyTap.ue;
        InputPort en = MyTap.tdrEn1;
        InputPort tck = tck;
        InputPort fso = MyTdr1.so;
    }
}

// instanced as chip.MyTap
Module tap {
    // ICL module read from source on or near line 1 of file './data/icl/
tap.icl'
    TCKPort tck;
    ScanInPort tdi;
    ScanOutPort tdo {
        Source IRMux;
    }
    TMSPort tms;
    TRSTPort trst;
    ToSelectPort tdrEn1 {
        Source sell1;
    }
    ScanInPort fromTdr1;
    ToSelectPort tapEn1 {
        Source IR[2];
    }
    ToCaptureEnPort ce;
    ToShiftEnPort se;
    ToUpdateEnPort ue;
    ScanInterface TAP {
        Port tdi;
        Port tdo;
        Port tms;
    }
    ScanInterface Internal {
        Port fromTdr1;
        Port tdo;
        Port tdrEn1;
```

```

        Port ce;
        Port se;
        Port ue;
    }
    ScanRegister IR[2:0] {
        ScanInSource tdi;
        CaptureSource 3'b001;
        ResetValue 3'b000;
    }
    ScanRegister bypass {
        ScanInSource tdi;
        CaptureSource 1'b0;
    }
    ScanMux IRMux SelectedBy fsm.irSel {
        1'b0 : DRMux;
        1'b1 : IR[0];
    }
    ScanMux DRMux SelectedBy IR[2:0] {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    }
    LogicSignal sel1 {
        IR[2:0] == 3'bx01;
    }
    Instance fsm Of tap1_fsm {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }
}

// instanced as tap.fsm
Module tap1_fsm {
    // ICL module read from source on or near line 59 of file './data/icl/
    tap.icl'
    TCKPort tck;
    TMSPort tms;
    TRSTPort trst;
    ToIRSelectPort irSel;
    ToResetPort tlr;
}

```

Related Topics

[get_icl_instances](#)
[read_icl](#)
[get_icl_modules](#)

report_iclock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the patterns -ijtag context, a pattern set has been opened by the open_pattern_set command.

Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.

Usage

report_iclock [-overridden]

Description

Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.

Arguments

- **-overridden**

Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

Return Values

None

Examples

The following example shows the result of the report_iclock command specified inside a pattern set in which two iClock commands have been issued. The first iClock command is on the reference pin of a PLL. The root-level clock port called CLK is its source and the connection is direct. The second one is on the CLK input of a memory BIST instance.

The report reflects the fact that the source tracing went through the PLL and picked up a FreqMultiplier value of 10 and a FreqDivider value of 2 either from the ICL definition or from an **iClockOverride** command that could have been specified on the output clock port of the PLL as shown in the example provided for the **iClock** command.

```
report_iclock
// iClock      Source  FreqMultiplier  FreqDivider      Period
// ======  =====  ======  ======  =====
// PLL.ref    CLK        1            1       100ns
// mbist.clk  CLK        10           2       20ns
```

Related Topics

[add_clocks](#)

[delete_clocks](#)
[get_iclock_list](#)
[get_iclock_option](#)
[iClock](#)
[iClockOverride](#)
[iRunLoop](#)
[report_clocks](#)

report_id_stamp

Context: dft -edt, patterns -scan

Mode: analysis

Displays the unique identifier that the tool assigns each internal pattern set.

Usage

`report_id_stamp [{> | >>} file_pathname]`

Description

Displays the unique identifier that the tool assigns each internal pattern set.

The report_id_stamp command displays the current internal pattern set's unique identification stamp which consists of five fields, each separated by a colon (:). Due to the length of three of the fields, the tool encodes those fields and displays the encoded information. The tool encodes the three fields using 4 bytes of hexadecimal numbers. This encoding's primary use is to ensure that each pattern set has a unique identification stamp. The following list shows the information each field provides:

1. Tool version number
2. Encoded environment settings
3. Encoded DRC rules data
4. Number of patterns in the internal pattern set
5. Encoded pattern data

The tool generates the identification stamp each time you issue either the report_id_stamp command or the write_patterns -Environment command. You can use the identification stamp to tag identical patterns saved in different formats.

Arguments

- `> file_pathname`

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- `>> file_pathname`

An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the unique identification stamp for the current pattern set:

`report_id_stamp`

2012.3:5c95:3e10:16:1bf2

Related Topics

[write_patterns](#)

report_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays current exceptions to Iddq restrictions.

Usage

```
report_iddq_exceptions [<> | >>] file_pathname]
```

Description

Displays current exceptions to Iddq restrictions.

The report_iddq_exceptions command displays Iddq exceptions added with the [add_iddq_exceptions](#) command. For more information about IDDQ, refer to “[IDDQ Test Set Creation](#)” in the *Scan and ATPG User’s Manual*.

Arguments

- `> file_pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- `>> file_pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays Iddq exceptions on sites for which the tool would report Iddq check violations if the exceptions did not exist.

```
report_iddq_exceptions
```

Related Topics

- [add_iddq_exceptions](#)
- [analyze_restrictions](#)
- [delete_iddq_exceptions](#)
- [report_gates](#)
- [report_primary_outputs](#)
- [set_iddq_checks](#)

report_ijtag_instances

Context: dft patterns

Mode: setup analysis insertion

Reports options of instances for which there is a matched ICL module.

Usage

```
report_ijtag_instances [name_spec] [-below_instances below_instances]
    [-filter filter] [-limit limit]
    [-silent]
```

Description

Reports options of instances for which there is a matched ICL module.

You can restrict the report to instances below a given collection of instances and filter the collection based on an attribute filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if there are no Ijtag instances to report.

Arguments

- *name_spec*

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all instances.

If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- *-below_instances instance_objects*

An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- *-filter attribute_equation*

An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on filtering attribute equation format.

- *-silent*

An optional switch that suppresses the warning message normally generated when the command does not find any instances to report. The warning message is intended to indicate

a probable mistake when specifying patterns. You can use the -silent switch to suppress the warning. This -silent option has no effect on any other error reporting.

Examples

The following example uses the report_ijtag_instances command to report the options of the first three instances of the Ijtag module. The command [create_dft_specification](#) provides access to those features if they are not already connected to the network.

report_ijtag_instances -l 3

```
//  
// ijtag instance: pb_di_do  
// -----  
// use_in_dft_specification : auto  
// scan_interface_count      : 1  
// data_in_port_count        : 4  
// data_out_port_count       : 0  
//  
// ijtag instance: pb_di_do/dido_bus  
// -----  
// use_in_dft_specification : auto  
// scan_interface_count      : 1  
// data_in_port_count        : 16  
// data_out_port_count       : 16  
//  
// ijtag instance: pb_si  
// -----  
// use_in_dft_specification : auto  
// scan_interface_count      : 1  
// data_in_port_count        : 0  
// data_out_port_count       : 0  
//  
// Reached limit of 3, skipping remaining 9 instances.
```

Related Topics

[get_ijtag_instances](#)
[set_ijtag_instance_options](#)
[get_ijtag_instance_option](#)

report_ijtag_logical_connections

Context: patterns -ijtag

Mode: setup, analysis

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports.

Usage

```
report_ijtag_logical_connections [-from source_pin_or_port_name]  
                                [-to destination_pin_or_port_name]
```

Description

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. If no pin or port connections are specified, all logical connections are listed.

Arguments

- **-from *source_pin_or_port_name***

An optional switch and value pair that specifies the source pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to *destination_pin_or_port_name***

An optional switch and value pair that specifies the destination pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

Examples

Example 1

The following example will list the logical connections between the pins /i1/in and /i32/u11/x10, as well as all logical connections from /i1/in and all connections to /i32/u11/x10.

```
report_ijtag_logical_connections -from /i1/in -to /i32/u11/x10  
// IJTAG Logical Connections:  
// From Source To Destination Inversion  
// ====== ====== ======  
// /i1/in /i32/u11/x10 no
```

Example 2

The following example will list all the logical connections in the current design.

```
add_ijtag_logical_connection -from tap1_I1/ue -to block3_I1/ue -inverted  
add_ijtag_logical_connection -from tap1_I1/ue -to logical_out -inverted
```

```
add_ijtag_logical_connection -from tdi -to tdo -inverted
add_ijtag_logical_connection -from tdi -to tms
add_ijtag_logical_connection -from tdo -to tms -inverted
report_ijtag_logical_connections

// IJTAG Logical Connections:
// From Source    To Destination   Inversion
// ======  ======  ======
// /tap1_I1/ue   /block3_I1/ue    yes
// /tap1_I1/ue   logical_out     yes
// tdi           tdo             yes
// tdi           tms             no
// tdo           tms             yes
```

Related Topics

[add_ijtag_logical_connection](#)
[delete_ijtag_logical_connection](#)

[report_ijtag_retargeting_options](#)

Context: all contexts

Mode: all modes

Reports the type and value of each option of the previous [set_ijtag_retargeting_options](#) command or their default values if the [set_ijtag_retargeting_options](#) command was not issued.

Usage

`report_ijtag_retargeting_options`

Description

Reports the type and value of each option of the previous [set_ijtag_retargeting_options](#) command or their default values if the [set_ijtag_retargeting_options](#) command was not issued.

Arguments

None

Examples

Example 1

The following example sets the context “patterns –ijtag” and reports the current retargeting options:

```
SETUP> set_context patterns -ijtag -no_rtl  
SETUP> report_ijtag_retargeting_options
```

Command Dictionary (R)
report_ijtag_retargeting_options

// Option	Type	Value
// test_setup_network_end_state	ENUM	reset
// test_setup_ireset	ENUM	when_icl_present
// iapply_target_annotations	ENUM	dense
// compare_constant_capture_values	ENUM	on
// compare_constant_data_out_ports	ENUM	off
// max_operations_per_iapply	INT	64
// merge_irunloop_only	BOOL	off
// tck_ratio	INT	auto
// inject_cycles	INT	0
// annotation_parameter_values	BOOL	on
// - user	ENUM	dense
// - iapply_targets	BOOL	on
// - scan_path_registers	BOOL	on
// - scan_load_unload_value	BOOL	on
// - tap_states	BOOL	on
// - network_end_state	BOOL	on
// - pragma_annotation	BOOL	on
// - pragma_bit_annotation	BOOL	on
// - pragma_variable	BOOL	on
// - pragma_bit_variable	BOOL	on
// - pragma_icl_checksum	BOOL	on
// - pragma_clock	BOOL	on
// - pragma_pattern_set	BOOL	on
// - pragma_tester_period	BOOL	on
// - pragma_expect_z	BOOL	on
// - svf_command_number	BOOL	on
// - icl_network_verify_activate	BOOL	on
// - icl_network_verify_flush	BOOL	on

Example 2

The following example sets the TCK ratio to 2, moves the tool to analysis mode, and reports the IJTAG options:

```
SETUP> set_ijtag_retargeting_options -tck_ratio 2
SETUP> set_system_mode analysis
...
ANALYSIS> report_ijtag_retargeting_options
```

```
Timeplate suitable for tck_ratio of 1:  
Timeplate tp1 =  
  Force_pi 0;  
  measure_po 0;  
  Pulse_clock 25 50;  
  period 100;  
end;  
  
Timeplate suitable for tck_ratio of 2;  
Timeplate tp2 =  
  force_pi 0;  
  measure_po 48;  
  pulse_clock 25 50;  
  force_tck 50;  
  period 100;  
end;  
  
Timeplate suitable for tck ratio of 4 or greater;  
Timeplate tp3 =  
  force_pi 0;  
  force_tck 0;  
  measure_po 96;  
  pulse_clock 25 50;  
  period 100;  
end;
```

Related Topics

[get_ijtag_retargeting_options](#)
[set_ijtag_retargeting_options](#)

report_input_constraints

Context: all contexts

Mode: setup, analysis

Displays the present constraint status of primary input pins.

Usage

Context: dft -scan and dft -test_points

```
report_input_constraints  
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs |  
-EQuivalent] [constraint_switch] [-Display {[FLAt_schematic] [HIEarchical_schematic]  
[DAta] | ALl} [APpend]]
```

Context: patterns -scan

```
report_input_constraints  
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs |  
-EQuivalent] [constraint_switch] [-HOLD] [-NO_Z] [-SLOW_pad [-CELL model_name]]  
[-Display {[FLAt_schematic] [HIEarchical_schematic] [DAta] | ALl} [APpend]]
```

Context: patterns -ijtag

```
report_input_constraints  
[-ALl | primary_input_pin_name... | -BIDI_Only | -BIDI_Exclude |  
-EQuivalent] [constraint_switch] [-Display {[FLAt_schematic] [HIEarchical_schematic]  
[DAta] | ALl} [APpend]]
```

Description

Displays the present constraint status of primary input pins.

The report_input_constraints command displays the constraint status of primary input pins to which you have applied constraints using the [add_input_constraints](#) command.

Arguments

- **-ALl**
An optional switch that specifies to display the current constraint status of every primary input pin to which you previously applied an input constraint using the [add_input_constraints](#) command. Pins from which you removed the constraint with the [delete_input_constraints](#) command prior to reporting the status are included in the report.
- ***primary_input_pin_name***
An optional, repeatable string that specifies the name of a primary input pin for which to display the constraint status.
- **-BIDI_Only**
An optional switch that specifies to display the constraint status of bidirectional pins only.

- **-BIDI_Exclude**
An optional switch that specifies to display the constraint status only of primary input pins that are not bidirectional.
- **-SCAN_INputs**
An optional switch that specifies to display the constraint status of scan input pins only.
- **-EQuivalent**
An optional switch that specifies to display the constraint status of primary input pins that have a pin equivalence definition applied to them. These are pins for which you used the -Equivalent switch with the add_input_constraints command. The -C_all constraint switch is the only constraint switch you can include in the same command line with -Equivalent.
- **constraint_switch**
An optional switch that specifies to report only on primary input pins that have a particular type of constraint applied. The choices, from which you can specify only one, are as follows:
 - C0 — Constant 0
 - C1 — Constant 1
 - CX — Constant X (unknown)
 - CZ — Constant Z (high-impedance)
 - CT0 — Constant TIE0
 - CT1 — Constant TIE1
 - CTZ — Constant TIEZ

Note

 For additional information about the constraints associated with the preceding switches, refer to the same switches under the [add_input_constraints](#) command.

- C_All** — All C-type constraints
- **-HOld**
An optional switch that specifies to report only input constraints that currently exhibit hold behavior. For information about what hold behavior is, refer to the -Hold argument under the add_input_constraints command.
- **-NO_Z**
An optional switch that specifies to report only input constraints that currently exhibit no-Z behavior. For information about what no-Z behavior is, refer to the -No_z argument under the add_input_constraints command.

- **-Slow_pad**

An optional switch that specifies to report only input constraints that currently exhibit slow-pad behavior. For information about what slow-pad behavior is, refer to the **-Slow_pad** argument under the **add_input_constraints** command.

- **-CELL *model_name***

An optional switch and string pair that, together with the **-Slow_pad** switch, specifies to prefix the instance name of each instance of the Tessent Cell library model, *model_name*, to the *primary_input_pin_name* and look up each resulting name as the pin on which to report.

- **-Display {[FLAt_schematic] [HIEarchical_schematic] [DAta] | ALl} [APpend]**

An optional switch, repeatable literal, and literal that specify to display the primary input(s) in the specified window. The “Append” option adds the primary inputs to the existing contents of the window instead of replacing the contents. The window choices are as follows:

FLAt_schematic — Specifies the Flat Schematic window. This is the invocation default.

HIEarchical_schematic — Specifies the Hierarchical Schematic window.

DAta — Specifies the Data window.

ALl — Specifies all supported windows (Flat Schematic, Hierarchical Schematic, and Data).

APpend — Adds the primary inputs to the existing contents of the window instead of replacing the contents.

Examples

The following example reports input constraints for a primary input.

report_input_constraints indata2

// Primary Input	C-Type	Hold	No-Z	Slow-Pad
// -----	-----	-----	-----	-----
// indata2	C0	No	Undetermined	No

Related Topics

[add_input_constraints](#)

[delete_input_constraints](#)

report_insert_test_logic_options

Context: dft -scan, dft -test_points

Mode: setup, analysis, insertion

Reports the prefix/infix values that will be used when logic is created by the insert_test_logic command.

Usage

```
report_insert_test_logic_options
```

Description

The report_insert_test_logic_options command reports the prefix/infix values that will be used when logic is created by the insert_test_logic command.

Arguments

None

Examples

This example reports the current prefix/infix values.

```
set_context dft -scan
report_insert_test_logic_options

SETUP> set_context dft -scan
SETUP> report_insert_test_logic_options
// 'insert_test_logic' Command Option      Current Value
// -----
// -inserted_object_prefix                ts_
// -logic_type clock_control_fix          clkfix_
// -logic_type clock_gater_fix            cgfix_
// -logic_type dedicated_input_holding_wrapper dihw_
// -logic_type dedicated_input_wrapper    diw_
// -logic_type dedicated_output_wrapper   dow_
// -logic_type generic                   logic_
// -logic_type lockup                   lockup_
// -logic_type pipeline                 pipeline_
// -logic_type ram_fix                  ramfix_
// -logic_type testpoint_control        cp_
// -logic_type testpoint_observation    op_
// -logic_type tri_state_fix           trifix_
// -logic_type x_bounding               xb_
// -persistent_cell_prefix             tesson_persistent_cell_
// -persistent_clock_cell_prefix       tesson_persistent_clock_cell_
```

Related Topics

[get_insert_test_logic_option](#)

[set_insert_test_logic_options](#)

report_iprocs

Context: all contexts

Mode: all modes

Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module.

Usage

report_iprocs

Description

Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module. The iProcNameSpace column is not displayed if all iProcs are defined in the global namespace.

Arguments

None

Examples

The following is an example of the report generated by this command:

```
report_iprocs
// Module  iProcNameSpace  iProcs
// =====
// chip          chipProc1  chipProc2  chipProc3  chipProc4
//           Bar      chipProc1  chipProc3
//           Foo      chipProc2  chipProc4
// block         blockProc1 blockProc2  blockProc3
//           Bar      blockProc4 blockProc5
//           Foo      blockProc1 blockProc3  blockProc5
//           Bar      blockProc2 blockProc4
```

report_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, lists the instance pathnames of a core stored in an instance-aware core-level LDB.

Usage

```
report_layout_core_information [-chip_design_name name]
```

Description

In hierarchical layout-aware diagnosis, lists the instance pathnames of a core stored in an instance-aware core-level LDB.

To use this command you must have a valid LDB that you previously opened using the open_layout command.

By default, this command lists the instance pathnames for all the designs stored in the LDB.

This command reports the pathnames, placement and orientation for all core instances for all top-level SoC designs stored in the currently opened LDB. It also reports the top-level DEF design name for the SoC design set in the current tool run.

See “[Diagnosis for Hierarchical Design](#)” in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- `-chip_design_name name`

An optional switch and string pair that returns only the instance pathnames for the specified design.

Examples

The following example shows the reporting format for the report_layout_core_information command.

```
report_layout_core_information
```

```
// Top design name: <top DEF design name-1>
// Instances of <core DEF design name>
//   '<Instance Path Name-1>' Placed at: <Placement-x>, <Placement-y>
//   Oriented: <Orientation>
//   '<Instance Path Name-2>' Placed at: <Placement-x>, <Placement-y>
//   Oriented: <Orientation>
//   ...
//
// Top design name: <top DEF design name-2>
// Instances of <core DEF design name>
//   '<Instance Path Name-1>' Placed at: <Placement-x>, <Placement-y>
//   Oriented: <Orientation>
//   '<Instance Path Name-2>' Placed at: <Placement-x>, <Placement-y>
//   Oriented: <Orientation>
//   ...
//
// Current design set to: <top DEF design name for current tool run>
```

Related Topics

[add_layout_core_information](#)

[delete_layout_core_information](#)

report_layout_files

Context: patterns -scan_diagnosis

Mode: setup, analysis

Returns the flat models and pattern sets associated with RCD constants stored in the LDB.

Usage

```
report_layout_files [-include_file_paths]
```

Arguments

- `-include_file_paths`

An optional switch that displays the full pathnames of the files.

Description

When generating RCD constants for RCD analysis, you may run the `create_feature_statistics` command multiple times on different combinations of flat models and pattern sets. Similarly, you may have verified the LDB against multiple flat models. The `report_layout_files` command reports which flat models and pattern sets are associated with the RCD constants stored in the LDB. It also reports the flat models that have been verified against the LDB.

The report displays the following information:

- `VERIFY_DB`: Within the LDB, the tool generates a verification database (.verifydb) after layout verification against a flat model.
- `RCD_DB`: Within the LDB, the tool generates RCD database (.rcddb) files every time you run `create_feature_statistics` to create RCD constants.
- `flat model`: The flat model name associated with `VERIFY_DB`.
- `flat MD5`: The MD5 signature for the flat model.
- `design MD5`: The structural MD5 of the design represented in the flat model.
- `pattern`: The pattern file name associated with `RCD_DB`. There may be multiple pattern files for each `RCD_DB`.
- `pattern MD5`: The MD5 signature for the pattern.
- `pattern set MD5`: The MD5 signature for the collective set of patterns in `RCD_DB`.

Examples

In the following example, the `report_layout_files` report shows that there are two verification databases within the current LDB. The first one is marked Active because the loaded flat model (current design) and pattern (current pattern) match its `RCD_DB` MD5 signatures.

```
read_flat_model design0.flat.gz
```

read_pattern pattern.wgl.gz

open_layout layoutDB

report_layout_files

RCD constants match for current design & pattern:

current design : design0.flat.gz MD5: c5ce18a2cfa9a98e3654a2d52b361c43
current pattern: pattern.wgl.gz MD5: 347b200939c92c0d82c2fdbd30a1074a9

Active	Type	Path/MD5
	VERIFY_DB	c5ce18a2cfa9a98e3654a2d52b361c43.verifydb
*****	RCD_DB	c5ce18a2cfa9a98e3654a2d52b361c43_da37.rcddb
	flat model	design0.flat.gz
	flat MD5	c5ce18a2cfa9a98e3654a2d52b361c43
	design MD5	7401fc0833c7c2c3169597dbf4d66566
	pattern	pattern.wgl.gz
	pattern	MD5347b200939c92c0d82c2fdbd30a1074a9
	pattern set MD5	da37e5d36ee09142ac4f3d4c4510dee2
	VERIFY_DB	54445c6b42d8237d88a4faa67a862abc.verifydb
	RCD_DB	54445c6b42d8237d88a4faa67a862abc_f522.rcddb
	flat model	design1.flat.gz
	flat MD5	54445c6b42d8237d88a4faa67a862abc
	design MD5	7401fc0833c7c2c3169597dbf4d66566
	pattern	pattern.wgl.gz
	pattern MD5	c15c47fa389a898d73337f2bb067152b
	pattern set MD5	f522b1857b12ed8af570486d041ddb34

report_layout_rules

Context: patterns -scan_diagnosis

Mode: analysis

Provides detailed reporting of layout rule violations.

Usage

```
report_layout_rules [-Fails_summary | -Summary | layout_rule_id... |  
layout_rule_id-occurrence#... | -All_fails | -MISmatch_report | -NO_LAYOUT_names]  
{[-INClude design_module_name...] | [-EXClude design_module_name...]}  
[{> | >>} file.pathname]
```

Description

Provides detailed reporting of layout rule violations. Before using this command, you must run layout verification as part of the [create_layout](#) command or use the [open_layout](#) command with an existing Layout Database.

Note

 Use this command in conjunction with the [create_layout](#) command to debug layout rule violations before you create the diagnosis Layout Database.

See “[Layout-Aware Diagnosis and Reporting](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

Arguments

- -Fails_summary

A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during layout verification:

- Rule identification (ID)
- Number of failures of the rule
- Current handling status of the rule
- Brief description of the rule

This is the default.

- -Summary

A switch that specifies to display a summary report of layout rule violations. When you report on all rules (report_layout_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during layout verification:

- Rule identification (ID)
- Number of failures of the rule

- Current handling status of the rule
- Brief description of the rule
- *layout_rule_id*
A repeatable string that specifies the identification literal (ID) of a particular layout rule for which you want to display the violations.
- *layout_rule_id-occurrence#*
A repeatable string that specifies the identification literal (ID) of a particular layout rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. Numbers to occurrences of layout rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.
- -All_Fails
A switch that specifies to display all occurrence messages for all occurrences of layout rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a “`report_layout_rules <layout_rule_id>`” command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.
- -Mismatch_report
An optional switch that re-generates the [layout verification rule violation report](#). Normally, you use this switch in conjunction with the `report_layout_rules` command’s -INClude and -EXClude switches to filter the layout rule violations—see “[Guidelines for Including or Excluding Design Modules From Mismatch Reporting](#)” in the *Tessent Diagnosis User’s Manual*.
- -NO_LAYOUT_names
An optional switch that suppresses the reporting of instance/net/pin path names in layout format for selected layout rule violations. See “[Instance, Net, and Pin Path Names in Layout Rule Violation Reports](#)” in the *Tessent Diagnosis User’s Manual* for complete information.
- -INClude *design_module_name*
A switch and repeatable string that defines an excluded area of the design’s hierarchy when performing layout-to-design verification. The excluded area comprises all design elements that are *not* below instances of the included modules.

The *design_module_name* is a space-delimited list of design modules. For example:

```
report_layout_rules -include moduleA moduleB moduleD
```

The -INClude and -EXClude switches are mutually exclusive.

For detailed usage of this switch, see “[Guidelines for Including or Excluding Design Modules From Mismatch Reporting](#)” in the *Tessent Diagnosis User’s Manual*.

- -EXClude **design_module_name**

A switch and repeatable string that defines an excluded area of the design's hierarchy when performing layout-to-design verification. The excluded area comprises all design elements contained below instances of the excluded modules.

The **design_module_name** is a space-delimited list of design modules. For example:

```
report_layout_rules -include moduleA moduleB moduleD
```

The -INClude and -EXClude switches are mutually exclusive.

For detailed usage of this switch, see “[Guidelines for Including or Excluding Design Modules From Mismatch Reporting](#)” in the *Tessent Diagnosis User's Manual*.

- >*file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file.pathname*.

- >>*file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

Example 1

report_layout_rules

```
// Warning: Rule DesignCellPinMatch violated 1 times out of 226 checks.  
// Warning: Rule DesignInstanceMatch violated 48 times out of 5344  
// checks.  
// Warning: Rule DesignNetPinMatch violated 139 times out of 18630 checks.  
// Warning: Rule LayoutNetPinMatch violated 66 times out of 18557 checks.  
// Warning: Rule PinExistenceMacro violated 3 times out of 44202 checks.
```

Without any switches a summary of all layout rule violations is printed. In addition to the number of violations, the total number of times a particular rule was checked is also printed. As an example, for the DesignInstanceMatch rule above, this rule was checked 5344 times (specifically there must be 5344 library cell instances in this design) and it was violated 48 out of these 5344 checks.

Example 2

report_layout_rules PinExistenceMacro

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist  
in macro).  
// Warning: PinExistenceMacro-1: The specified NET cpu_i/uOPD/nx104 is  
// connected to MACRO oai222 which uses a PIN C0 which is not defined in  
// the LEF files.  
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is  
// connected to MACRO oai222 which uses a PIN C0 which is not defined in  
// the LEF files.  
// Warning: PinExistenceMacro-3: The specified NET cpu_i/INTR_PRI1_0_ is  
// connected to MACRO oai222 which uses a PIN C0 which is not defined in  
// the LEF files.
```

By specifying a rule name all the violations of that rule will be printed. First a summary for the user specified rule is printed and then each rule violation is printed in detail.

Example 3

report_layout_rules PinExistenceMacro-2

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist  
in macro).  
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is  
// connected to MACRO oai222 which uses a PIN C0 which is not defined in  
// the LEF files.
```

This prints one particular violation of a given rule.

Example 4

In this case the instance name in DFT format differs from that in layout format since the last part of the name is flattened. Since the layout is being created from the LEF/DEF the instance path name in layout format is expected to be as follows:

```
/core0/cntl_1/mac\adder\AND0
```

Note that in LEF/DEF naming convention individual hierarchy delimiters are escaped.

Report the corresponding Instance Rules rule violation using the following command:

report_layout_rules DesignInstanceMatch-1

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0  
// (Expected Layout Name: /core0/cntl_1/mac\adder\AND0) not found in  
// layout.
```

You can suppress reporting of the layout name for this violation using the report_layout_rules command as follows:

report_layout_rules DesignInstanceMatch-1 -no_layout_names

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0 not  
// found in layout.
```

Related Topics

[create_layout](#)

[open_layout](#)

report_lbist_configuration

Context: dft -logic_bist

Mode: setup, analysis

Reports the global LogicBIST controller configuration parameters.

Usage

```
report_lbist_configuration [> | >>file_pathname] [-hardware_default_compatibility]
```

Description

Reports the global LogicBIST controller configuration parameters. To report the parameters for individual EDT/LogicBist blocks, use the [report_edt_configurations](#) command. This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- `>file_pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- `>>file_pathname`
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.
- `-hardware_default_compatibility`
An optional switch that specifies to report hardware default configuration values. This switch is only available for fault simulation. When used in setup mode, it reports the hardware default values. When used in analysis mode, it reports these values plus fault simulation settings and compatibility on a per setting and composite basis. The tool issues a warning if you specify this switch outside of the TCD flow.

Examples

Example 1

The following example shows how to configure the LBIST controller parameters and then report on what has been specified.

```
set_lbist_controller_options -max_shift 400 -max_capture 3 -max_pattern 100000 \
    -capture {clk1P 30 clk2P 60 rstP 10}
report_lbist_configuration
```

```
// IP version: 1
// Max Shift cycles: 400
// Max Capture cycles: 3
// Max pattern count: 100000
// Capture procedures: clk1P 30%//
// clk2P 60%
// rstP 10%
```

Example 2

The following snippet shows the report format that the tool returns when you specify the `-hardware_default_compatibility` switch while in analysis mode. The report indicates that some settings are compatible with hardware default mode and others are not.

```
// -----
// Core Instance
// -----
// LBIST
// -----
// shift length      56          40      no
// capture width    4           15      no
// pattern count    500         1024    yes
// masked chains    0           0       yes
// NCP 1 count     154 (60%)   103 (40%) no
// NCP 2 count     26 (10%)    102 (40%) no
// NCP 3 count     76 (30%)    51 (20%) no
// ...
// ...
// -----
// Current fault simulation settings are not compatible with hardware default mode
```

Related Topics

[report_edt_configurations](#)

[set_lbist_power_controller_options](#)

report_lbist_pins

Context: dft -logic_bist

Mode: setup, analysis

Reports the LogicBIST controller pins using the set_lbist_pins command.

Usage

```
report_lbist_pins [> | >> file_pathname]
```

Description

Reports the LogicBIST controller pins using the set_lbist_pins command.

When you have renamed pins using the set_lbist_pins -on_controller_module command, the report includes a column entitled “Name on LBIST module.” By default, this column is hidden.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User’s Manual](#) for complete information.

Arguments

- *>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- *>>file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

This example reports all the design pins to which the LogicBIST controller’s control signals, including the TAP interface signals are connected.

```
report_lbist_pins

//          PinType           Connection
//  -----   -----
//  Clock      refclk
//  Scan enable scan_en
//  TCK        tck
//  Xbounding enable lbist_en
//  Setup Shift Scan in tdi
//  Setup Shift Scan out tdo jtag/scanCfgReg_so
//  Shift DR    - jtag/shift_dr
//  Capture DR  - jtag/capture_dr
//  Update DR   - jtag/update_dr
//  Test logic reset - jtag/tlr
//  TAP instruction decode - jtag/scanCfgReg_en
```

Related Topics

[set_lbist_pins](#)

[report_edt_pins](#)

report_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.

Usage

```
report_lfsr_connections
```

Description

Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.

The report_lfsr_connections command displays all of the connections between the LFSRs and the primary pins. These connections were specified with the add_lfsr_connections commands.

Arguments

None

Examples

The following example displays the connections between the LFSRs and primary pins:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs misr1 misr 5 11 -parallel -in
add_lfsr_taps lfsr1 1 3
add_lfsr_taps lfsr2 2 3
add_lfsr_connections scan_in.1 lfsr1 3
add_lfsr_connections scan_out.0 misr1 2
report_lfsr_connections
```

Related Topics

[add_lfsr_connections](#)

[delete_lfsr_connections](#)

report_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).

Usage

```
report_lfsrs
```

Description

Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).

The report_lfsrs command displays all of the LFSRs with their current values and tap positions that you specified using the add_lfsrs and add_lfsr_taps commands.

Arguments

None

Examples

Example 1

The following example displays the definitions of all the current LFSRs:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs misr1 misr 5 11 -parallel -in
report_lfsrs
```

Example 2

The following example defines a 24-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, and 23 and then lists the LFSRs:

```
add_lfsrs my_prpg prpg 24 FFFFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
report_lfsrs
```

Output:

```
list of LFSRs
my_prpg  PRPG  length=24  shift=serial  ca_taps=3,12,15,23
my_prpg_pre_unload_value = 111111111111111111111111
my_prpg_post_unload_value = 111111111111111111111111
```

Related Topics

[add_lfsrs](#)

[add_lfsr_taps](#)

[delete_lfsrs](#)

[set_lfsrs](#)

report_licenses

Context: unspecified, all contexts

Mode: all modes

Reports the license features currently checked out by the tool.

Usage

`report_licenses`

Description

Reports the license features currently checked out by the tool.

Arguments

None

Examples

The following example displays the licenses currently being used by the automatic diagnosis session.

`report_licenses`

The following report displays:

```
Licenses:  
4 yieldascandiag  
1 yieldaylearn
```

report_lists

Context: dft -edt, patterns -scan

Mode: analysis

Displays the list of pins whose values the tool will report during simulation.

Usage

report_lists [**>** | **>>**] *file_pathname*

Description

Displays the list of pins whose values the tool will report during simulation.

The **report_lists** command displays pins the tool is currently set up to report on during simulation. Use the [add_lists](#) command to add pins to the list and the [delete_lists](#) command to remove pins from the list.

When you switch to Setup mode, the tool discards all pins from the list.

Arguments

- **> *file_pathname***
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the pins whose values the tool will report during simulation:

```
add_lists /i_1006/o /i_1007/o
report_lists

2 pins are currently monitored.
i_1006/o
i_1007/o
```

Related Topics

- [add_lists](#)
- [delete_lists](#)
- [set_list_file](#)

report_loadboard_loopback_pairs

Context: patterns -ijtag, -scan, -scan_diagnosis, -scan_retargeting

Modes: Setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Reports the loopback connections and the ac_delay, if any, that were created using the add_loadboard_loopback_pairs command.

Usage

```
report_loadboard_loopback_pairs
```

Description

This command reports any loopback connection pairs created using the [add_loadboard_loopback_pairs](#) command. It will also report the ac_delay for the loopback pairs.

Arguments

None

Examples

This example shows the report of loopback pairs and ac_delay values for the elaborated design:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]}
```

```
report_loadboard_loopback_pairs
```

```
-----
Inputs   Outputs   AC Time to Z
-----  -----
din[4]   dout [5]  N/A
ue       dout [0]  N/A
-----
```

report_loops

Context: all contexts

Mode: setup, analysis

Displays information about circuit loops.

Usage

```
report_loops [-All | loop_id#...]  
[-Display {HIERarchical_schematic [-Hierarchical]} [DAta] | ALl} [APpend] ]  
[{> | >>} file.pathname]
```

Description

Displays information about circuit loops.

The report_loops command displays information about currently identified loops in the circuit. For each loop, the report indicates whether the loop was broken by duplication. The report shows loops unbroken by duplication to be broken instead by a constant value, which means the loop is either a coupling loop or has a single multiple-fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

You can write the loops report information to a file by using the command's redirection operators or the [write_loops](#) command.

Arguments

- **-ALl**
An optional switch that specifies to report all the loops in the circuit. This is the default.
- ***loop_id#***
An optional, repeatable, positive integer that specifies the identification number of a particular loop to report. The tool assigns loop identification numbers consecutively, starting with 1.
- **-Display {HIERarchical_schematic [-Hierarchical]} [DAta] | ALl} [APpend]**
An optional switch, repeatable literal, and literal that specify to display the primary input(s) in the specified window; if *Append* is specified, the primary inputs are added to the existing contents of the window instead of replacing the contents.

The choices are as follows:

HIERarchical_schematic — A literal that specifies the Hierarchical Schematic window. If the *-Hierarchical* literal is specified, only the top-level hierarchical instances are displayed in the Hierarchical Schematic window; if the *-Hierarchical* literal is not specified, the child instances of the corresponding hierarchical instances are also displayed.

DAta— A literal that specifies the Data window.

All — A literal that displays the information in all the preceding windows.

- `>file.pathname`
An optional redirection operator and pathname pair for creating *or* replacing the contents of `file.pathname`.
- `>>file.pathname`
An optional redirection operator and pathname pair for appending to the contents of `file.pathname`.

Examples

Example 1

The following example displays a list of all the loops in the circuit:

```
set_system_mode analysis
report_loops

Loop = 1: not_duplicated (coupling loop)
    my_design/my_minibus (SBUS)
    my_design/PAD (BUF)
    my_design/my_minibus (Z2X)
Loop = 2: not_duplicated (coupling loop)
...
Loop = 8: not_duplicated (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 2

The following example displays loop 8 only:

```
report_loops 8

Loop = 8: not_duplicated (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 3

The following example writes the display information for loop 8 to a new file named `my_loop_file`:

```
report_loops 8 > my_loop_file
... writing to file my_loop_file
```

Related Topics

[report_feedback_paths](#)
[write_loops](#)

report_lpct_configuration

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports the current low pin count test controller (LPCT) parameters to the standard display output.

Usage

`report_lpct_configuration [> | >> filename]`

Description

Reports the current low pin count test controller (LPCT) parameters to the standard display output.

After saving the patterns, you can use the [report_register_value](#) command to report the different fields of the test configuration registers such as shift_length, capture_depth, scan_pattern_count, and chain_pattern_count.

For more details on the values reported by this command, see the [set_lpct_controller](#) command.

Arguments

- `> file_pathname`
An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of *file_pathname* with command output.
- `>> file_pathname`
An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of *file_pathname*.

Examples

The following example shows a report generated from Setup mode during EDT logic creation:

```
report_lpct_configuration
// Controller type:           Type 3
// Max Shift cycles:          1000
// Max Capture cycles:        8
// Max Scan patterns:         1048575
// Max Chain patterns:        1023
// Test mode detect sequence: 1010000101001
// Test mode detect period:   100 cycles
// Scan shift control:        internally generated clock
```

Related Topics

[report_lpct_pins](#)

[set_lpct_controller](#)

report_lpct_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.

Usage

```
report_lpct_pins [{> | >>} filename]
```

Description

Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.

Arguments

- *> file_pathname*
An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of *file_pathname* with command output.
- *>> file_pathname*
An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of *file_pathname*.

Examples

The following example shows a report generated from Setup mode during EDT logic creation:

```
report_lpct_pins

// Input pin      Pin name      Inversion Internal connection
// -----        -----
// Clock          port_clock    -
// Reset (Active High)   -        -      pwr_up/reset
// Data in        lpct_data_in  -
// Test mode      lpct_test_mode -
// 
// Output pin     Connection
// -----        -----
// Capture enable
// Clock mux select
// Shift clock
// Scan enable    scan_en
// Reset out
// Test end
// Data out
// Test active
```

Related Topics

[report_lpct_configuration](#)
[set_lpct_controller](#)

report_measure_cycles

Prerequisites: A WGL or STIL pattern set must be loaded.

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.

Usage

```
report_measure_cycles [-All_cycles] [-Suite test_suite_ID] [{-Pattern_range | -Cycle_range}  
begin end] | [-List cycles ...] [observe_point...] [> | >> filename]
```

Description

Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.

Depending on the arguments used with this command, you can report expect bits, masked bits, cycle padding, cycles, patterns, and chain cells and lengths.

You can use this command to troubleshoot data verification errors that occur before diagnosis. Only WGL and STIL test patterns are supported by this command.

Use this command to debug failure file mismatches reported by the tool. You can use this command to report the failing cycle, failing pattern, expect values, and boundary cycles related to a specific mismatch.

Arguments

- **-All_cycles**

An optional switch that generates a full report for all cycles.

The tool assigns each cycle to a specific category, and groups the neighboring cycles belonging to the same category into a cycle block, which include one or more cycles.

For each cycle block, the tool reports the following in columnar format:

- **First** — First cycle.
- **Last** — Last cycle (unless the cycle is the same as the first cycle).
- **#Cycle** — Count of cycles.
- **Pid** — Pattern ID.
- **Type** — Pattern type.
- **Sid** — Suite ID. Reported if there is more than one test suite.
- **RelPid** — Relative Pattern ID. Reported if there is more than one test suite.

- **Operation** — Function of the cycle block, for example “Test setup”, “Setup shift”, “Force PI”, “Measure PO” and “Shift and measure SCO”.
- **-Suite** *test_suite_ID*
An optional switch and string pair that specifies the targeted test suite when you use more than one test suite. When you specify a valid test suite ID, the pattern IDs (begin end) specified are treated as relative IDs for this targeted test suite. Otherwise, the pattern IDs (begin end) are considered as absolute numbers.
By default, if you omit this switch, then the pattern IDs (begin end) are considered as absolute numbers. If only the -suite switch is used and the -pattern is omitted, then the tool reports all patterns of the specified test suite.
- **-Pattern_range** *begin end*
Optional switch and value pair that specifies a range of patterns.
- **-Cycle_range** *begin end*
Optional switch and value pair that specifies a range of cycles. This option cannot be used with -All_cycles because a full cycle-oriented report is generated; the tool ignores the -all_cycles option, if supplied.
- **-List** *cycles*
Optional, repeatable switch and value pair used to specify one or more specific cycles.
- **observe_point**
Optional value that specifies the name of one or more components to focus the report on. Observe points are usually pin names, but may also represent any of the following:
 - chain
 - scanout_pin
 - edt_channel
 - edt_scanout_pin
 - POpin
- **>** *file_pathname*
An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*. Use this argument to write the report to a file.
- **>>** *file_pathname*
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*. Use this argument to append the report to the specified file.

Examples

Example 1

The following example displays a specified range of patterns:

```
report_measure_cycles -pattern 10 20
```

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern   Measure_PO   Measure_SCO (first,last,step)
  -----  -----
  10 (S)    2277    { (2279,2380,+1) }
  11 (S)    2381    { (2383,2484,+1) }
  12 (S)    2485    { (2487,2588,+1) }
  13 (S)    2589    { (2591,2692,+1) }
  14 (S)    2693    { (2695,2796,+1) }
  15 (S)    2797    { (2799,2900,+1) }
  16 (S)    2901    { (2903,3004,+1) }
  17 (S)    3005    { (3007,3108,+1) }
  18 (S)    3109    { (3111,3212,+1) }
  19 (S)    3213    { (3215,3316,+1) }
  20 (S)    3317    { (3319,3420,+1) }
```

Example 2

The following example displays the cycles, patterns, and expect bits information for the chain test for chain3:

```
report_measure_cycles chain3
```

```
total cycles = 41103
cycle  pattern  chain{cell}:scanout  expect
-----  -----
 105      0      chain3{0}:scan_out3 L  chaintest
 106      0      chain3{1}:scan_out3 L  chaintest
 107      0      chain3{2}:scan_out3 H  chaintest
 108      0      chain3{3}:scan_out3 H  chaintest
 109      0      chain3{4}:scan_out3 L  chaintest
 110      0      chain3{5}:scan_out3 L  chaintest
 111      0      chain3{6}:scan_out3 H  chaintest
 112      0      chain3{7}:scan_out3 H  chaintest
 113      0      chain3{8}:scan_out3 L  chaintest
 114      0      chain3{9}:scan_out3 L  chaintest
 115      0      chain3{10}:scan_out3 H  chaintest
 116      0      chain3{11}:scan_out3 H  chaintest
 117      0      chain3{12}:scan_out3 L  chaintest
 118      0      chain3{13}:scan_out3 L  chaintest
 119      0      chain3{14}:scan_out3 H  chaintest
 120      0      chain3{15}:scan_out3 H  chaintest
```

Example 3

The following example reports all cycles based on cycle blocks.

report_measure_cycles -all_cycles

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
```

First	Last	#Cycle	Pid	Type	Operation
0		1	0	C	Test setup
1		1	0	C	Setup shift
2	103	102	0	C	Shift (load only)
104		1	0	C	Setup shift
105	206	102	0	C	Shift and measure SCO
207		1	1	C	Setup shift
208	309	102	1	C	Shift and measure SCO
310		1	2	C	Setup shift
311	412	102	2	C	Shift and measure SCO
413		1	3	C	Setup shift
414	515	102	3	C	Shift and measure SCO
516		1	4	C	Setup shift
517	618	102	4	C	Shift and measure SCO
619		1	5	C	Setup shift
620	721	102	5	C	Shift and measure SCO
722		1	6	C	Setup shift
723	824	102	6	C	Shift and measure SCO
825		1	7	C	Setup shift
826	927	102	7	C	Shift and measure SCO
928		1	8	C	Setup shift
929	1030	102	8	C	Shift and measure SCO
1031		1	9	C	Setup shift
1032	1133	102	9	C	Shift and measure SCO
1134		1	0	S	Setup shift
1135	1236	102	0	S	Shift (load only)
1237		1	0	S	Measure PO and pulse clock
1238		1	0	S	Setup shift
1239	1340	102	0	S	Shift and measure SCO
1341		1	1	S	Measure PO and pulse clock

Example 4

The following example reports the measurement cycles for test suite 1.

report_measure_cycles -suite 1

```

total cycles = 82206
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
pattern set # 2 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern (Suite/Pid)  Measure_PO  Measure_SCO (first,last,step)
  -----
  0 ( 1/ 0) (C)  ---- { (105,206,+1) }
  1 ( 1/ 1) (C)  ---- { (208,309,+1) }
  2 ( 1/ 2) (C)  ---- { (311,412,+1) }
  3 ( 1/ 3) (C)  ---- { (414,515,+1) }
  4 ( 1/ 4) (C)  ---- { (517,618,+1) }
  5 ( 1/ 5) (C)  ---- { (620,721,+1) }
  6 ( 1/ 6) (C)  ---- { (723,824,+1) }
  7 ( 1/ 7) (C)  ---- { (826,927,+1) }
  8 ( 1/ 8) (C)  ---- { (929,1030,+1) }
  9 ( 1/ 9) (C)  ---- { (1032,1133,+1) }
  0 ( 1/ 0) (S) 1237 { (1239,1340,+1) }
  1 ( 1/ 1) (S) 1341 { (1343,1444,+1) }
  2 ( 1/ 2) (S) 1445 { (1447,1548,+1) }
  3 ( 1/ 3) (S) 1549 { (1551,1652,+1) }
  4 ( 1/ 4) (S) 1653 { (1655,1756,+1) }
  5 ( 1/ 5) (S) 1757 { (1759,1860,+1) }
  6 ( 1/ 6) (S) 1861 { (1863,1964,+1) }

```

Example 5

The following example reports in blocks, all cycles between 0 and 200, and shows that cycles 105-206 are measure cycles for the first chain pattern.

report_measure_cycles -cycle_range 0 200

```

total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )

First  Last   #Cycle  Pid  Type  Operation
  0        1      0    C  Test setup
  1        1      0    C  Setup shift
  2     103    102    0    C  Shift (load only)
  104      1      0    C  Setup shift
  105    206    102    0    C  Shift and measure SCO

```

Related Topics

[diagnose_failures](#)
[read_patterns](#)
[set_diagnosis_options](#)

report_memory_cluster_configuration

Context: dft, patterns

Mode: all modes

Reports the configuration of cluster memories tested in subsequent controller steps for a specified memory cluster and memory access level.

Usage

Context: dft

```
report_memory_cluster_configuration
{ [-design_name] [design_name]
  [-design_id] [design_id]
  [-controller_id] [controller_id]
  [-cluster_id] [cluster_id] }
| { [-tcd_memory_cluster_name] [cluster_module_name]
    [-memory_access_level] {from_defaults_specification|logical|auto|physical} } }
```

Context: patterns

```
report_memory_cluster_configuration
{ [-design_name] [design_name]
  [-design_id] [design_id]
  [-pattern_id] [pattern_id] }
| { [-patterns_name] [patterns_name]
    [-test_step_name] [test_step_name]
    [-controller_icl_instance] [controller_icl_instance] }
| [-tcd_memory_bist_controller_name] [tcd_memory_bist_controller_name]
```

Description

Creates a tabular report with steps and lists of cluster memories tested in consecutive controller steps. In the dft context, the report is based on the memory cluster library description and the selected memory expansion (access) level. In the patterns context, the report is based on the MemoryBIST controller TCD description associated with the given MemoryBIST controller instance.

Context : dft

The command accepts a minimum set of options to uniquely identify the reported memory cluster instance in either the [DftSpecification](#) or the [/Core/MemoryCluster](#), in case the DftSpecification was not loaded. When a design contains a single memory cluster, it is not necessary to specify any options with the command. When multiple memory clusters exist in a design, an error is reported if it is not possible to uniquely identify a memory cluster_id with the provided command options. In this case, the DftSpecification can be consulted with the [report_config_data](#) command to identify the available memory clusters

If the reported memory cluster is included in the DftSpecification, it is accompanied with the memory expansion level value defined either explicitly with the [/DftSpecification/MemoryBist/MemoryCluster/memory_access_level](#) property, or inferred from the [/DefaultsSpecification/DftSpecification](#) settings.

If no DftSpecification is present, Tesson Shell reports the memory cluster configuration of the [/Core/MemoryCluster](#) defined in the tcd partition. Similarly to the DftSpecification generation, the cluster memory access level is assumed from the DefaultsSpecification, unless the “-memory_access_level” option is used.

Context : patterns

The command accepts a minimum set of options to uniquely identify a MemoryBIST controller TCD loaded in the tcd partition. Alternately, you may directly specify a controller TCD using the “-tcd_memory_bist_controller_name” option. When a design contains a single memory cluster, it is not necessary to specify any options with the command. When multiple memory clusters exist in a design, an error is reported if it is not possible to uniquely identify a memory cluster_id with the provided command options. In this case, the DftSpecification can be consulted with the [report_config_data](#) command to identify the available memory clusters.

Arguments

- **-design_name *design_name***

An optional switch and value pair that specifies which DftSpecification or PatternsSpecification wrapper the reported memory cluster is located in. This is used in case multiple DftSpecifications or PatternsSpecifications for different design names are loaded in memory. In the dft context, this switch is mutually exclusive with the “-tcd_memory_cluster_name” switch. In the patterns context, this switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch.

- **-design_id *design_id***

An optional switch and value pair that narrows down the search of the DftSpecification or PatternsSpecification when multiple DftSpecifications or PatternsSpecifications are loaded in memory for the same design. In the dft context, this switch is mutually exclusive with the “-tcd_memory_cluster_name” switch. In the patterns context, this switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch.

- **-controller_id *controller_id***

An optional switch and value pair to narrow down the search to only a single DftSpecification/MemoryBist/Controller wrapper containing the reported memory cluster. This switch is mutually exclusive with the “-tcd_memory_cluster_name” switch.

- **-cluster_id *cluster_id***

An optional switch and value pair that informs which /DftSpecification/MemoryBist/Controller/MemoryCluster to select for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_cluster_name” switch.

- **-tcd_memory_cluster_name *cluster_module_name***
An optional switch and value pair to reference the reported memory cluster in /Core/MemoryCluster. This switch is mutually exclusive with the following switches referencing DftSpecification location: “-design_name”, “-design_id”, “-controller_id”, “-cluster_id”.
- **-memory_access_level {from_defaults_specification | logical | auto | physical}**
An optional switch and enumeration symbol pair used together with the “-tcd_memory_cluster_name” switch to assume the given cluster memory expansion level when generating the report. If the switch is not provided, Tesson Shell assumes the expansion level currently defined in the /DefaultSpecification/DftSpecification.
- **-pattern_id *pattern_id***
An optional switch and value pair to specify a particular PatternsSpecification to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.
- **-patterns_name *patterns_name***
An optional switch and value pair to specify a particular /PatternsSpecification/Patterns wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.
- **-test_step_name *test_step_name***
An optional switch and value pair to specify a particular /PatternsSpecification/Patterns/TestStep wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.
- **-controller_icl_instance *controller_icl_instance***
An optional switch and value pair to specify a particular /PatternsSpecification/Patterns/TestStep/MemoryBist/Controller wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.
- **-tcd_memory_bist_controller_name *tcd_memory_bist_controller_name***
An optional switch and value pair to reference the reported memory controller in the /Core/MemoryBistController configuration data. This switch only available in the patterns context, and is mutually exclusive with the following switches referencing PatternsSpecification location: “-patterns_name”, “-pattern_id”, “-test_step_name”, and “-controller_icl_instance”.

Examples

Example 1 (Context: dft)

Given the following DftSpecification:

```
DftSpecification(WIRELESS_CORE,rtl) {
    MemoryBist {
        Controller(c1) {
            MemoryCluster(c1) {
                instance_name: top/a/b/mc;
                memory_access_level: physical;
            }
        }
    }
}
```

The following command can be used to report the test scheduling for memory cluster instance top/a/b/mc:

```
report_memory_cluster_configuration -design_name WIRELESS_CORE -controller_id c1
```

Note

 The cluster will be tested at the physical memory access level in this example.

Example 2 (Context: dft)

Assuming no design is loaded, but the following DftSpecification exists (no_insertion mode):

```
DftSpecification(WIRELESS_CORE,rtl) {
    MemoryBist {
        Controller(c1) {
            MemoryCluster(c1) {
                memory_cluster_library_name: CLUSTER;
            }
        }
    }
}
```

The same command given in Example 1 can be used to print the test scheduling. Please note however, this time the memory access level defined in the /DefaultsSpecification/ DftSpecification applies.

Example 3 (Context: dft)

If the DftSpecification is not available, one may print the test scheduling with the following command:

```
report_memory_cluster_configuration -tcd_memory_cluster_name CLUSTER \
-memroy_access_level physical
```

Example 4 (Context: patterns)

Given the following PatternsSpecification loaded in memory:

```
PatternsSpecification(WIRELESS_CORE,rtl,signoff) {
    Patterns(ICLNetwork) {
        ICLNetworkVerify(WIRELESS_CORE) {
        }
    }
    Patterns(MemoryBist_P1) {
        TestStep(run_time_prog) {
            MemoryBist {
                run_mode : run_time_prog;
                reduced_address_count : off;
                Controller(mbist_c1_controller_inst) {
                    DiagnosisOptions {
                        compare_go : on;
                        compare_go_id : on;
                    }
                }
            }
            Controller(mbist_c2_shared_bus_assembly_inst.mbist_c2_controller_inst) {
                DiagnosisOptions {
                    compare_go : on;
                    compare_go_id : on;
                }
            }
        }
    }
}
```

The following command invocation can be used to print the report for the second controller:

```
report_memory_cluster_configuration -controller_icl_instance \
mbist_c2_shared_bus_assembly_inst.mbist_c2_controller_inst
```

Example 5 (Context: patterns)

If the controller TCD is loaded in the tcd partition in memory, the report can be printed with the following command:

```
SETUP> report_memory_cluster_configuration \
-tcd_memory_bist_controller_name mbist_c2_controller
```

This command results in the following table report:

```
// controller id: mbist_c2_controller
// =====
// Step  Access  Logical  Physical  Cluster   Logical  Physical  Memory
//      level    access    access    interface  memory   memory   id
//           code     code     id
// -----
// 0    logical  3'b001   -        I1       LM_0
// 1    logical  3'b010   -        I1       LM_1
// 2    logical  3'b011   -        I1       LM_2
// 3    logical  3'b100   -        I1       LM_3
//
```

report_memory_identification

Context: dft

Mode: analysis, insertion

Reports the validation results of the memory module matching performed from the name patterns specified with the set_memory_identification_options command.

Usage

report_memory_identification

Description

Creates a tabular report showing the memory module matching results. It is required to have previously used the [set_memory_identification_options](#) command with a valid match expression before requesting the report created by report_memory_identification.

The report contains a Memory Identification Report table with the following information:

- **Design module** — The name of the memory design module identified either by the pattern(s) specified with the set_memory_identification_options command, or recognized by the tool by matching a provided memory TCD.
- **TCD name** — If found, the name of the memory TCD associated with the design memory module is reported.
- **Pattern match** — A “yes” or “no” report as to whether a design memory module matches any pattern specified with the set_memory_identification_options command.
- **Instance count** — The number of instances associated with a design memory module.
- **Action** — Recommended actions for the user to review. The reported actions are:
 - **missing pattern** — The listed design module has not been identified by a matching pattern expression specified with the set_memory_identification_options command, but has been recognized as memory because the memory TCD for the module exists. The pattern matching expression specified for set_memory_identification_options needs to be verified.
 - **missing TCD** — The listed design module has been identified as memory by a matching pattern expression specified with the set_memory_identification_options command, but the corresponding memory TCD has not been found. The memory TCD needs to be provided, or the pattern expression needs to be updated if the design module is not a memory module.

The report also contains a Matching Patterns Expressions table with the following information:

- **Matching pattern expression** — The regular or wildcard matching pattern specified with the set_memory_identification_options command.

- **Matched modules count** — The number of design modules that are matched by the corresponding regular or wildcard matching pattern.

Arguments

None

Examples

During the setup mode for this example design, the following design module matching patterns are specified:

```
set_memory_identification_options "{(.SYNC_1.*)} {(.SYNC_2.*)} {(.SYNC_3.*)}" -regexp
```

When [check_design_rules](#) is run, the report will automatically be generated if errors or warning exist between design module matching and memory TCD.

Later in the flow, while in either the analysis or insertion mode, the [report_memory_identification](#) command can be run to view the report as shown in the example below:

```
report_memory_identification

// Memory identification report table
// =====
// Design module      TCD name    Pattern match   Instance count   Action
// -----      -----
// SYNC_1R1W_16x8     -          yes            5               missing TCD
// SYNC_2R2W_12x8     -          yes            3               missing TCD
//
// Actions legend:
//
// Action 'missing TCD':
//   Memory has been identified by matching pattern expression, but
//   corresponding memory TCD library has not been found. Provide the
//   missing TCD library file or correct the matching pattern expression
//   if identified design module is not a memory module.

// Matching pattern expressions table
// =====
// Matching pattern expression  Matched modules count
// -----
// (.SYNC_1.*)
// (.SYNC_2.*)
// (.SYNC_3.*)
//
// Error: Memory validation by matching pattern expression (enabled with
//        set_memory_identification_options) has identified potentially
//        untested memories due to missing TCD library file.
//        Refer to the above memory identification report table.
```

Related Topics

[set_memory_identification_options](#)

report_memory_instances

Context: dft, patterns

Mode: all modes

Prerequisite: The current design must be set with the `set_current_design` command.

Reports the options of all instances for which there is a matched TCD Memory description.

Usage

```
report_memory_instances [name_patterns] [-below_instances below_instances]  
[-filter filter] [-limit limit]  
[-silent]
```

Description

Reports the options of all instances for which there is a matched TCD Memory description. See “[Tessent Core Description](#)” on page 3755.

You can restrict the report to instances below a given collection of instances, and filter the collection based on an attribute filtering equation involving other attributes registered against or inherited by the instance object type.

The `-silent` option suppresses the warning that is normally generated if there are no memory instances to report.

Arguments

- `name_patterns`

An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces `{}`. If no `name_patterns` are specified, the tool searches all instances.

If you specify the `-below_instances` options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise, the name patterns are relative to the current design.

- `-below_instances instance_objects`

An optional switch and value pair that constrains the command to search for instances below the specified `instance_objects`. The specified `name_patterns` are searched relative to each instance found in the `instance_objects` list.

- `-filter attribute_equation`

An optional switch and string pair that specifies to filter results based on the expression specified by the `attribute_equation` string. The attributes used within the equation must exist for the instance object type. See “[Attribute Filtering Equation Syntax](#)” on page 3162 for more details on the attribute filtering equation format.

- **-limit *limit***

An optional switch and number pair that defines the maximum number of memory instances the tool should report. The default of limit is 10.

- **-silent**

An optional switch that suppresses the warning message normally generated when the command does not find any instances to report. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to suppress the warning. This -silent option has no effect on any other error reporting.

Examples

The following example uses the report_memory_instances command to report the options of the first three instances of the memory module.

SETUP> report_memory_instances -limit 3

```
//  
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem1  
// -----  
// bist_data_in_pipelineing : off  
// physical_cluster_override :  
// power_domain_island :  
// test_clock_override :  
// use_in_memory_bist_dft_specification : auto  
// use_in_memory_bisr_dft_specification : auto  
//  
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem2  
// -----  
// bist_data_in_pipelineing : off  
// physical_cluster_override :  
// power_domain_island :  
// test_clock_override :  
// use_in_memory_bist_dft_specification : auto  
// use_in_memory_bisr_dft_specification : auto  
//  
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem3  
// -----  
// bist_data_in_pipelineing : off  
// physical_cluster_override :  
// power_domain_island :  
// test_clock_override :  
// use_in_memory_bist_dft_specification : auto  
// use_in_memory_bisr_dft_specification : auto  
//  
// Reached limit of 3, skipping remaining 15 instances.
```

Related Topics

[get_memory_instances](#)
[set_memory_instance_options](#)
[get_memory_instance_option](#)

report_memory_repair_groups

Context: dft

Mode: setup, analysis

Reports information about the repair groups from a DftSpecification.

Usage

```
report_memory_repair_groups [-show_memory_instances]
    {[-design_name design_name -insertion_id id]}
```

Description

The report_memory_repair_groups command reports information about the repair groups from a DftSpecification. By default, the command reports the repair groups from the DftSpecification for the current elaborated design and insertion id.

If it is desired to report the repair groups from a DftSpecification of a different design than the one that is currently elaborated, or if there isn't one currently elaborated, the -design_name and -insertion_id arguments must be specified.

The report contains the controller for each group and its size, expressed in terms of total memory bits that are sharing a spare element. If the -show_memory_instances argument is specified, the report will also provide an additional list of all memory instances within each repair group.

Arguments

- **-show_memory_instances**
An optional switch that specifies a list of all memory instances with each repair group will also be reported.
- **-design_name *design_name***
An optional switch and value pair that specifies the module design name from which the repair groups will be reported. The -insertion_id argument must also be specified to uniquely identify the design module.
- **-insertion_id *id***
An optional switch and value pair that specifies the module id associated with the design module from which the repair groups will be reported. The -design_name argument must also be specified to uniquely identify the design module.

Examples

The following example shows the reporting of repair group information from the DftSpecification for the currently elaborated design. The use of the -show_memory_instances argument provides the additional list of memory instances within each repair group.

```
report_memory_repair_groups -show_memory_instances
```

Command Dictionary (R)
report_memory_repair_groups

```
Repair Group Memory Instances
=====
-----  -----  -----
Controller Repair Group Size
-----  -----  -----
c1        RG0      3.0kilobits
c2        RG0      512.0bits
c2        RG1      576.0bits
c3        RG0      1.0kilobits
c4        RG0      512.0bits
c5        RG0      256.0bits
c5        RG1      1.0kilobits
c6        RG0      1.0kilobits
c6        RG1      256.0bits
```

```
Repair Group Memory Instances
=====
-----  -----  -----
Controller Repair Group Memory Instance
-----  -----  -----
c1        RG0      MEM00
c1        RG0      MEM01
c2        RG0      \mem@10
c2        RG1      mem1
c3        RG0      mem2
c3        RG0      mem3
c4        RG0      mem4
c4        RG0      mem5
c5        RG0      mem6
c5        RG1      mem7
c6        RG0      mem8
c6        RG1      mem9
```

report_mismatch_sources

Context: dft -edt, patterns -scan

Mode: analysis

Displays the sources of simulation mismatches.

Usage

```
report_mismatch_sources [src-id | -All] [-Display [FLAt_schematic]  
[HIEarchical_schematic] [DAta] [Wave] | All] [(> | >>) file_pathname]
```

Description

Displays the sources of simulation mismatches.

The report_mismatch_sources command displays detailed information about the sources of simulation mismatches found by a previous analyze_simulation_mismatches command or a write_patterns -Debug command. For each mismatch source you specify, the command reports the following information in columnar format:

- ID# — Positive integer that identifies a specific mismatch source. The tool numbers mismatch sources consecutively, starting at 1.
- Instance name — Hierarchical instance name of the mismatch source.
- Pattern number — The number of the first failing pattern caused by a mismatch source. In the ModelSim Wave window, the pattern number is the value of the “_pattern_count” variable at the time the mismatch occurred. This variable is displayed in the top line of the Wave window and increments as each pattern is simulated. This number corresponds to the pattern number in ASCII patterns.
- Time (ns) — First simulation time where the source causes a value in the external simulator that is inconsistent with what the tool expected.
- Capture cycle — The capture cycle where the first mismatch occurs (starting from 0).
- Expected value (ATPG_val) — Expected output value from the tool.
- Simulated value (VCD_val) — Simulated output value from the external simulator. Possible values are:
 - 0 — Logic 0
 - 1 — Logic 1
 - Z — Tristate
 - X — Unknown
 - U — Unused
 - . — No VCD data was saved.

- Reason — The following is a list of possible mismatch reasons:
 - “incorrect input”: Mismatch is due to wrong force values at PI.
 - “incorrect PO measure”: Mismatch is due to wrong measure PO time.
 - “incorrect loading”: Mismatch is due to incorrect scan load.
 - “incorrect capture”: Mismatch is due to incorrect capture clock even at the state element.
 - “incorrect unloading”: Mismatch is due to incorrect scan unload.
 - “wrong clock pulse”: Mismatch is due to incorrect clock pulse.
 - “setup time violation”: Mismatch is due to setup time violation, specifically the combinational path is not stable in one clock cycle.
 - “incorrect logic”: Mismatch is due to difference logic between Tesson Cell library model and the Verilog model. This includes different netlists used between ATPG tool and Verilog simulator, or Tesson Cell library model difference to the corresponding Verilog model.
 - “mismatched RAM model”: Mismatch is due to difference on RAM model between Tesson Cell library and Verilog library.
 - “incorrect library cell”: Mismatch is due to difference between Tesson Cell library model and Verilog library cell.
 - “incorrect ATPG setting”: Mismatch is due to improper setting on ATPG tool. Most common mistakes are incorrect setting of set_split_capture and set_clock_off_simulation.
 - “hold time violation”: Mismatch is due to hold time violation. The sink flop of the hold time violation is reported as the mismatch source gate.
 - “clock/data port race”: mismatch is due to clock and data racing. Common clock racing situations include C6 and C4 violation.

Arguments

- *src-id*
An optional positive integer that specifies the identification number of the mismatch source to report. The tool assigns source IDs to mismatch sources consecutively, starting with 1.
- -Display [FLAt_schematic] [HIEarchical_schematic] [DAta] [Wave]
An optional switch that specifies to display the specified source(s) in the Flat Schematic, Hierarchical Schematic, Data, or Wave window of DFTVisualizer. If the *Display* switch is specified without an argument, the Flat Schematic, Data, and Wave windows are displayed.
- -All
An optional switch that specifies to display all the mismatch sources. This is the default.

Note

 In the Flat Schematic window, pin names assigned to the value “.” indicate that the VCD debug test bench did not capture the VCD value for that location. It is not possible to capture VCD values for every node in the design due to very large file sizes and run time; the test bench only captures values in the combination cone behind the failure location, which in most cases provides sufficient information to explain the mismatch.

- *>file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file.pathname*.

- *>>file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

The following example assumes the pathname of the ModelSim bin directory is defined by the PATH variable and patterns are generated but not yet saved. The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates the Verilog library *my_mismatch_work*, compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. Finally, DFTVisualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

```
set_external_simulator vsim -c -voptargs+=acc=npr
write_patterns results pat_p.v -verilog -parallel -replace
sys vlib my_mismatch_work
sys vlog -work my_mismatch_work my_design_source/my_design.v -v \
    my_verilog_source/verilog_lib.v

Model Technology ModelSim SE vlog 5.5a Compiler 2001.04
-- Compiling module DFFS_scan
-- Compiling module my_top_level_module
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling module xor3
-- Compiling module mux21
...
-- Compiling module buff
-- Compiling module mgc_latch
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling UDP ud_dff
-- Compiling UDP ud_dlat

Top level modules: my_top_level_module

analyze_simulation_mismatches -auto -internal -lib my_mismatch_work
```

Command Dictionary (R) **report_mismatch_sources**

```
// New directory work_dft_debug has been created to store files for mismatch debug.  
// Previous simulation mismatch sources are deleted.  
//-----  
//Step-1 Creating Verilog testbench work_dft_debug/mentor_default.v with failure file  
//dump.  
//-----  
//Step-2 Running Verilog testbench work_dft_debug/mentor_default.v to create failure file  
//-----  
//Step-3 Loading failure file work_dft_debug/mentor_default.v.fail.  
//-----  
//Step-4 Creating Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v with VCD dump  
//-----  
//Step-5 Running Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v to create VCD  
//file.  
//Running ModelSim to create VCD file work_dft_debug/mentor_default.v_debug.vcd  
//-----  
//Step-6 Loading VCD file work_dft_debug/mentor_default.v debug.vcd.  
//-----  
//Step-7 Analyzing 25 internal scan patterns in parallel format.  
//-----  
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason  
//--- -----  
// 1 /ix1286/Y (364)      scan-1       1900          0     1     0   incorrect lib  
cell "and4"  
// 2 /ix1278/Y (265)      scan-15      17300          0     1     0   incorrect lib  
cell "and4"
```

report_mismatch_sources 2 -display flat_schematic data

DFTVisualizer opens and displays the following information about mismatches in the Transcript window; the sources are also displayed in the Flat Schematic and Data windows.

```
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason  
//--- -----  
// 2 /ix1278/Y (265)      scan-15      17300          0     1     0   incorrect lib  
//                                         cell "and4"
```

For more information about the Flat Schematic and Data windows, refer to “[DFTVisualizer Window Overview](#)” in the *Tessent Shell User’s Manual*.

Related Topics

[analyze_simulation_mismatches](#)
[write_patterns](#)
[set_external_simulator](#)

report_misrs

Context: pattern -scan

Mode: setup, analysis

Reports the MISRs you have added to your design.

Usage

```
report_misrs {-all | misr_name ...} [> | >> file_pathname]
```

Description

Reports the MISRs you have added to your design.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- **-all**
A required switch that reports all MISRs.
- **misr_name**
A required repeatable string that specifies the name of the MISR.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

This example reports all the MISRs defined across all the EDT/logicBIST blocks.

```
report_misrs -all
```

```
list of MISRs
A_misr_0  MISR  length=24  in_stream_taps=20,21,23
  A_misr_0_pre_unload_value = 00000000000000000000000000000000
A_misr_1  MISR  length=32  in_stream_taps=4,5,31
  A_misr_1_pre_unload_value = 00000000000000000000000000000000
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 000000000001000100010001000100000000
A_misr_3  MISR  length=32  in_stream_taps=4,5,31
  A_misr_3_pre_unload_value = 00000000000000000000000000000000
B1_misr  MISR  length=24  in_stream_taps=20,21,23
  B1_misr_pre_unload_value = 00000000000000000000000000000000
B2_misr_0  MISR  length=24  in_stream_taps=20,21,23
  B2_misr_0_pre_unload_value = 00000000000000000000000000000000
B2_misr_1  MISR  length=24  in_stream_taps=20,21,23
  B2_misr_1_pre_unload_value = 00000000000000000000000000000000
```

Example 2

This example reports a specific MISR: A_misr_2:

```
report_misrs A_misr_2
```

```
list of MISRs
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 00000000000100010001000100000000
```

Related Topics

[add_misrs](#)

[delete_misrs](#)

report_misr_connections

Context: patterns -scan

Mode: setup, analysis

Reports MISR connections.

Usage

```
report_misr_connections misr_name [> | >>file.pathname]
```

Description

Reports MISR connections.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- **misr_name**
A required repeatable string that specifies the name of the MISR.
- **>file.pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.
- **>>file.pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

This example shows the connection between the LBIST scan chain output and the MISR input for a 24-bit MISR named 'misr'. The 16 scan chains are compacted with a 4-to-1 XOR and connected to the first 4 inputs of the MISR.

```
report_misr_connections misr
```

```
list of MISR connections
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0] MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1] MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2] MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3] MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4] MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5] MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6] MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7] MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8] MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9] MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10] MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11] MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12] MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13] MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14] MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15] MISR=misr
connection_points=4
```

Related Topics

[add_misrs](#)

[set_misr_connections](#)

report_module_matching

Context: dft, patterns -ijtag

Mode: all modes

Reports design modules and their matching modules of the specified format, if the set_current_design command has already been executed.

Usage

```
report_module_matching -icl | -format { icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory  
| tcd_memory_cluster }
```

Description

Reports design modules and their matching modules of the specified format, if the set_current_design command has already been executed.

Arguments

- -icl

An optional switch that specifies to report matching ICL modules.

If the attribute “ignore_during_icl_extraction” is defined for any instance, an additional column “Ignored” is added to the report specifying whether the instance was ignored during ICL extraction. However, the “Ignored” column is not included if the value of the attribute is set to “false” for all instances for which the attribute is defined.

- - format { icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster }

An optional switch that specifies the format of the matching modules to be reported.

Return Values

None

Examples

Example 1

The following example shows an example report generated by this command.

```
read_icl ./data/icl/block3.icl  
set_design_sources -format icl -search_design_load_path On  
set_module_matching_options -suffix_pattern_list "\[0-9\]" -regexp  
set_current_design chip  
report_module_matching -icl
```

Command Dictionary (R)
report_module_matching

Design Module	Design Instance	ICL Module	ICL File
PLL_CORE	chip/pll1_I1/iPLL	PLL_CORE	./data/icl_prim/PLL_CORE.icl
block1_0	chip/block1_I1	block1	./data/icl/block1.icl
block1_1	chip/block1_I2	block1	./data/icl/block1.icl
block2	chip/block2_I1	block2	./data/icl/block2.icl
block3	chip/block3_I1	block3	./data/icl/block3.icl
pll1	chip/pll1_I1	pll1	./data/icl_prim/pll1.icl
raw1_0	chip/block1_I1/raw1_I1	raw1	./data/icl_prim/raw1.icl
raw1_1	chip/block3_I1/raw1_I4	raw1	./data/icl_prim/raw1.icl
raw1_2	chip/block3_I1/raw1_I3	raw1	./data/icl_prim/raw1.icl
raw1_3	chip/block3_I1/raw1_I2	raw1	./data/icl_prim/raw1.icl
raw1_4	chip/block3_I1/raw1_I1	raw1	./data/icl_prim/raw1.icl
raw1_5	chip/block2_I1/raw1_I2	raw1	./data/icl_prim/raw1.icl
raw1_6	chip/block2_I1/raw1_I1	raw1	./data/icl_prim/raw1.icl
raw1_7	chip/block1_I2/raw1_I2	raw1	./data/icl_prim/raw1.icl
raw1_8	chip/block1_I2/raw1_I1	raw1	./data/icl_prim/raw1.icl
raw1_9	chip/block1_I1/raw1_I2	raw1	./data/icl_prim/raw1.icl
sib1_0	chip/sib1_IPLL	sib1	./data/icl_prim/sib1.icl
sib1_1	chip/block1_I2/sib1_I2	sib1	./data/icl_prim/sib1.icl
sib1_2	chip/block1_I2/sib1_I1	sib1	./data/icl_prim/sib1.icl
sib1_3	chip/block1_I1/sib1_I2	sib1	./data/icl_prim/sib1.icl
sib1_4	chip/block1_I1/sib1_I1	sib1	./data/icl_prim/sib1.icl
sib1_5	chip/sib1_I4	sib1	./data/icl_prim/sib1.icl
sib1_6	chip/sib1_I3	sib1	./data/icl_prim/sib1.icl
sib1_7	chip/sib1_I2	sib1	./data/icl_prim/sib1.icl
sib1_8	chip/sib1_I1	sib1	./data/icl_prim/sib1.icl
tap1	chip/tap1_I1	tap1	./data/icl_prim/tap1.icl
tdr1	chip/pll1_I1/tdr1_IPLL1	tdr1	./data/icl_prim/tdr1.icl
tdr1_1	chip/block3_I1/tdr1_I4	tdr1	./data/icl_prim/tdr1.icl
tdr1_2	chip/block3_I1/tdr1_I3	tdr1	./data/icl_prim/tdr1.icl
tdr1_3	chip/block3_I1/tdr1_I2	tdr1	./data/icl_prim/tdr1.icl
tdr1_4	chip/block3_I1/tdr1_I1	tdr1	./data/icl_prim/tdr1.icl
tdr1_5	chip/block1_I2/tdr1_I2	tdr1	./data/icl_prim/tdr1.icl
tdr1_6	chip/block1_I2/tdr1_I1	tdr1	./data/icl_prim/tdr1.icl
tdr1_7	chip/block1_I1/tdr1_I2	tdr1	./data/icl_prim/tdr1.icl
tdr1_8	chip/block1_I1/tdr1_I1	tdr1	./data/icl_prim/tdr1.icl
tdr2	chip/block2_I1/tdr2_I1	tdr2	./data/icl_prim/tdr2.icl

Example 2

In the following example, the “ignore_during_icl_extraction” attribute is defined for three instances which are identified in the “Ignored” column in the report generated by the report_module_matching command.

```
set_attribute_value chip/sib1_I3 -name ignore_during_icl_extraction -value false
set_attribute_value chip/sib1_I2 -name ignore_during_icl_extraction -value true
set_attribute_value chip/sib1_I1 -name ignore_during_icl_extraction -value true
report_module_matching -icl
```

The following is a snippet of the generated report:

Design Module	Design Instance	Ignored	ICL Module	ICL File
sib1_5	chip/sib1_I4	No	sib1	./data/icl_prim/sib1.icl
sib1_6	chip/sib1_I3	No	sib1	./data/icl_prim/sib1.icl
sib1_7	chip/sib1_I2	Yes	sib1	./data/icl_prim/sib1.icl
sib1_8	chip/sib1_I1	Yes	sib1	./data/icl_prim/sib1.icl

Example 3

The following example shows an example report of matching tcd_memory_cluster modules.

`report_module_matching -format tcd_memory_cluster`

The following is a snippet of the generated report:

Design Module	Design Instance	Module	File
A2Mem1	A1Inst1/A2InstMem1	A2Mem1	./data/A2Mem1.tcd_mem_lib_cluster
A2Mem1	A1Inst1/A2InstMem2	A2Mem1	./data/A2Mem1.tcd_mem_lib_cluster
B4Mem1	B1Inst1/B2Inst1/B4InstMem1	B4Mem1	./data/B4Mem1.tcd_mem_lib_cluster
B4Mem1	B1Inst1/B2Inst1/B3Inst1/B4InstMem1	B4Mem1	./data/B4Mem1.tcd_mem_lib_cluster
B4Mem2	B1Inst1/B2Inst1/B3Inst1/B4InstMem2	B4Mem2	./data/B4Mem2.tcd_mem_lib_cluster
B4Mem4	B1Inst1/B2Inst1/B3Inst1/B4InstMem4	B4Mem4	./data/B4Mem4.tcd_mem_lib_cluster

Related Topics

[set_module_matching_options](#)

[report_module_matching_options](#)

[set_design_sources](#)

[set_current_design](#)

report_module_matching_options

Context: all contexts

Mode: all modes

Reports the current settings defined by the set_module_matching_options command.

Usage

```
report_module_matching_options
```

Description

Reports the current settings defined by the [set_module_matching_options](#) command.

Arguments

None

Return Values

None

Examples

Example 1

The following example shows an example report generated by this command.

```
SETUP> set_design_sources -format icl -search_design_load_path On
SETUP> set_module_matching_options -suffix_pattern_list "_mod_[0-9]" -regexp
SETUP> set_module_matching_options -prefix_pattern_list mgc -append
SETUP> set_module_matching_options -suffix_pattern_list "\[0-9]" -regexp -append
SETUP> report_module_matching_options

// Option           Value
// -----          -----
// case insensitivity override  off
//
// affix type   regexp  pattern
// -----  -----  -----
// prefix      no      "mgc"
// suffix       yes     "_mod_[0-9]"
// suffix       yes     "\[0-9]"
```

Example 2

This example shows the output of the report_module_matching_options command after you have specified the following command and switch:

```
set_insertion_options -edited_module_prefix edited_module_prefix
```

Note the “by insertion*” report column and the addendum at the bottom. This will only be visible when you have specified the prefix using the aforementioned command and switch.

```
SETUP> set_module_matching_options -prefix_pattern_list {mentor} -nocase -append
SETUP> set_insertion_options -edited_module_prefix {www_}
SETUP> report_module_matching_options

// Option                                Value
// -----  -----
// case insensitivity override  on
//
// affix type   regexp   pattern      by insertion*
// -----  -----  -----  -----
// prefix      no       " "          no
// prefix      no       "mentor"    no
// prefix      no       "www_"     yes
// suffix      yes      "[_] + [0-9] +" no
// by insertion*: Additional prefix for ICL format defined by command
// 'set_insertion_options -edited_module_prefix'
```

Related Topics

[set_module_matching_options](#)

report_multicycle_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays multicycle path definitions you previously read into the tool using the [read_sdc](#) command.

Usage

```
report_multicycle_paths [-All] [-Error] [-Verbose] [-Debug_error]  
[-From source_node... | -FROM_Clock clock_signal_name...]  
[-TO sink_node... | -TO_Clock clock_signal_name...]  
[-Through through_node...]...  
[{> | >>} file.pathname]
```

Description

Displays multicycle path definitions you previously read into the tool using the [read_sdc](#) command.

Note that you can display SDC filenames and line numbers by first issuing the “[set_timing_exceptions_handling](#) -line_no” command. To analyze if a pin or a gate is affected by a multicycle path, you can use “[set_gate_report](#) -timing_exceptions on” and [report_gates](#) commands.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) multicycle paths are reported.

Arguments

Tip When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- **-All**
An optional switch that displays all previously-defined multicycle paths. This is the default.
- **-Error**
An optional switch that displays only the multicycle paths that have errors and warnings.
- **-Verbose**
An optional switch that displays all the gate information in the from, to, and through gate lists.

- **-Debug_error**
An optional switch that displays all the previously-defined multicycle paths that have errors and warnings, along with their respective error and warning messages.
- **-From *source_node*...**
An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the output pins of the instance are considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.
- **-FROM_Clock *clock_signal_name*...**
An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.
- **-TO *sink_node*...**
An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

Note

 For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_multicycle_paths... -to CLK2” would report multicycle paths with an end point at D2.

- **-TO_Clock *clock_signal_name*...**
An optional switch and repeatable string pair that specifies the end point of a multicycle paths by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal and the data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.
- **{-THrough *through_node*...}...**
An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance

pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, their order left-to-right in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following multicycle path specification:

```
report multicycle paths -from A -through B C -through D  
E -to F
```

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be multicycle paths.

- *>file_pathname*
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- *>>file_pathname*
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays all previously-defined multicycle paths.

```
report_multicycle_paths -all
```

```
// Multicycle Path -from -from /cpu/d_in[7] -through /cpu/u1 -cycles 2
// Multicycle Path -to /cpu/c -cycles 2
//
// -----
// The multicycle path below has an error.
// -----
// Multicycle Path -from dummy1 -to dummy2 -cycles 2
//
// -----
// The multicycle paths below have warnings.
// -----
// Multicycle Path -from /cpu/d_in[7] -through /cpu/d_in_[6]
//   /cpu/u25 -cycles 2
// Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
// Multicycle Path -from CLK1 -through /cpu/xray[6] CLK1 -cycles 2
//
// Total reported multicycle paths = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//           Use the command "report_multicycle_paths -debug_error" to
//           report the causes of the errors/warnings.
// Warning: Multicycle path defined through all primary inputs, including
//           all clocks, which prohibits at-speed testing. Deleted clocks
//           from this list.
// Warning: There exists a multicycle path that prohibits at-speed testing
//           of clock domain /CLK1.
//           This may result in lower test coverage.
```

Example 2

The following example displays the multicycle paths with error and warning messages.

report_multicycle_paths -error

```
// Error: Multicycle Path -from dummy1 -to dummy2 -cycles 2
// Warning: Multicycle Path -through /cpu/d_in_[6] /cpu/u25 -cycles 2
// Warning: Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
// Warning: Multicycle Path -from CLK1 -through /cpu/xray[6]
//   CLK1 -cycles 2
// Total reported multicycle paths = 4
```

Example 3

The following example displays debug information for the multicycle paths with error and warning messages.

report_multicycle_paths -debug_error

```
// Error: Multicycle Path -from dummy1 -to dummy2 -cycles 2 did not find
//         gates dummy1 dummy2
// Warning: Multicycle Path -from /cpu/d_in[7]-through /cpu/d_in[6]
//           /cpu/u25 -cycles 2 has no combinational path between start and
//           through points.
// Warning: Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
//           defined through all primary inputs, including all clocks,
//           which prohibits at-speed testing.
//           Deleted clocks from this list.
// Warning: Multicycle Path -from CLK1 -through /cpu/xray[6] CLK1
//           -cycles 2 prohibits at-speed testing of clock domain /CLK1.
// Multicycle path debug summary: 1 error, 3 warnings.
```

Example 4

The following example displays a verbose report of multicycle paths that start from /cpu/d_in[7]. Notice the empty “()” for the incorrectly defined path.

```
report_multicycle_paths -verbose -from /cpu/d_in[7]

// Multicycle Path -from /cpu/d_in[7] (/reg_d_1_7_/Q (81)) -through
//   /cpu/u1 (/reg_d_1_6_/Q (73)) -cycles 2
//
// -----
// The multicycle path below has an error.
// -----
// Error: Multicycle Path -from /cpu/d_in[7] ( ) -through /cpu/xray[6] ( )
//   -cycles 2
//
// Total reported false paths = 2
```

For examples of how to use the asterisk (*) character to specify multiple pathnames with one pathname string, see the [add_false_paths](#) command.

Example 5

The following shows multicycle path reporting after turning on reporting of SDC filenames and line numbers:

```
read_sdc sdc2
set_timing_exceptions_handling -line_no on
report_multicycle_paths

// Multicycle Path -from sff108/Q -to sff209/D -cycles 3 -setup
//   (filename = sdc2, line 1)
```

Related Topics

[add_false_paths](#)
[delete_false_paths](#)
[delete_multicycle_paths](#)
[read_sdc](#)
[report_false_paths](#)

set_timing_exceptions_handling

report_multiprocessing_options

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Displays the values of all variables used when executing multiprocessing commands for ATPG or fault simulation.

Usage

`report_multiprocessing_options`

Description

Displays the values of all variables used when executing multiprocessing commands for ATPG or fault simulation. The `report_multiprocessing_options` command displays in human-readable format the values of multiprocessing variables that affect the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the `set_multiprocessing_options` command. To obtain the values of multiprocessing processing variables for Tcl scripting, use the `get_multiprocessing_option` command.

For more information, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual* and “[The Tessent Tcl Interface](#)” in the *Tessent Shell User’s Manual*.

Arguments

None

Examples

The following example displays the current settings of the multiprocessing variables:

`report_multiprocessing_options`

// Multiprocessingoptions:// Option	Type	Value	Description
// -----			
// generic_delete	string		generic job scheduler delete
// generic_scheduler	string		generic job scheduler
// license_timeout	number	5	A positive integer to specify in minutes how long to queue for a license, 'unlimited', or 'no-queue'
//			
//			
// lsf_heuristics	on/off	on	true for LSF job scheduler heuristic learning
// lsf_learning	on/off	on	true for LSF job scheduler learning
// lsf_options	string		options for LSF job scheduler
// multithreading	on/off	on	turn on/off multithreading flow
// processors_per_grid_request	number	-1	# processors grouped for one grid request (default: 1 for SGE and GENERIC, 4 for LSF)
//			
// remote_shell	string	rsh	rsh or ssh remote_shell setting
// result_time_limit	float	45	time limit (min) used to detect non-responsive slaves.
//			
// scheduler_timeout	number	10	# mins for job scheduler
// sge_options	string		options for SGE job scheduler

Related Topics

[add_processors](#)
[delete_processors](#)
[get_multiprocessing_option](#)
[report_processors](#)
[set_multiprocessing_options](#)

report_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the nofault settings for the specified pin pathnames or pin names of instances.

Usage

```
report_nofaults pathname... | -All [-Instance] [-Stuck_at {01 | 0 | 1}] [-Class {Full | User | System}] [{> | >>} file_pathname]
```

Description

Displays the nofault settings for the specified pin pathnames or pin names of instances.

The report_nofaults command displays, for pin pathnames or pin names of instances, the nofault settings that you previously specified with the add_nofaults command.

Arguments

- ***pathname***

A repeatable string that specifies the pin pathnames or the instance pathnames for which you want to display the nofault settings. If you specify an instance pathname, you must also specify the -Instance switch.

- ***-All***

A switch that displays the nofault settings on either all pin pathnames or, if you also specify the -Instance switch, all pin names of instances.

- ***-Instance***

An optional switch that specifies that the *pathname* or -All argument indicates instance pathnames.

- ***-Stuck_at* 01 | 0 | 1**

An optional switch and literal pair that specifies the stuck-at or transition nofault settings you want to display. The choices are as follows:

01 — A literal that specifies to display both the “stuck-at-0” and “stuck-at-1” nofault settings for stuck-at faults; or both the “slow-to-rise” and “slow-to-fall” nofault settings for transition faults. This is the default.

0 — A literal that specifies to display only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

1 — A literal that specifies to display only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- -Class Full | User | System

An optional switch and literal pair that specifies the source (or class) of the nofault settings which you want to display. The literal choices are as follows:

Full — A literal that displays all the nofault settings in the user and system class. This is the default.

User — A literal that displays only the user-entered nofault settings.

System — A literal that displays only the netlist-described nofault settings.

- $>file_pathname$

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.

- $>>file_pathname$

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

The following example displays all pin names of the instances that have the nofault settings:

```
add_nofaults i_1006 i_1007 i_1008 -instance  
report_nofaults
```

Related Topics

[add_faults](#)

[add_synchronous_clock_group](#)

[delete_faults](#)

[delete_nofaults](#)

[report_faults](#)

report_nonscan_cells

Context: dft -edt, patterns -scan

Mode: analysis

Displays the non-scan cells whose model type you specify.

Usage

```
report_nonscan_cells [-All | -TIE0 | -TIE1 | -TIEX | -TLatch | -Clock_sequential |  
-INIT0 | -INIT1] [-VERilog] [{> | >>} file_pathname]
```

Description

The report_nonscan_cells command displays the instance name, gate number, and model type of the non-scan cells that you specify. The tool assigns a model type to the non-scan cells during the Design Rules Check (DRC) in order to model non-scan memory behavior more exactly.

During scan loading, scan clocks can affect non-scan memory elements. The worst case is to assume that all non-scan memory elements will have an unknown state right after scan loading, which makes some faults difficult to test. Therefore, the tool assigns an appropriate model type to the non-scan cells, which sets them to known values. The following argument descriptions describe the conditions that the tool uses to make the various model type assignments.

Arguments

- **-All**
An optional switch that displays all non-scan cells. This is the default.
- **-TIE0**
An optional switch that displays the non-scan cells which are always 0 after each loading and before the next unloading.
Non-scan cells that the tool models as TIE0 indicate that the pin constraints hold the cell's value during non-scan operation.
- **-TIE1**
An optional switch that displays the non-scan cells which are always 1 after each loading and before the next unloading.
Non-scan cells that the tool models as TIE1 indicate that the pin constraints hold the cell's value during non-scan operation.
- **-TIEX**
An optional switch that displays the non-scan cells which are always unknown (X) after each loading and before the next unloading.
- **-TLatch**
An optional switch that displays the non-scan cells which the tool models as transparent latches.

- **-Clock_sequential**
An optional switch that displays all clock_sequential cells that are valid when the sequential depth is set to non-zero.
- **-INIT0**
An optional switch that displays a list of cells which DRC has identified as being initialized to 0 by scan load but has failed to prove that it is tied to 0.
Non-scan cells that the tool models as INIT0 indicate one of the following:
 - The test procedure can set the memory element to 0 after scan loading.
 - The non-off clock is a reset.
- **-INIT1**
An optional switch that displays a list of cells which DRC has identified as being initialized to 1 by scan load, but has failed to prove that it is tied to 1.
Non-scan cells that the tool models as INIT1 indicate one of the following:
 - The test procedure can set the memory element to 1 after scan loading.
 - The non-off clock is a set.
- **-VERilog**
An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example reports only the non-scan cells that the tool models as transparent latches.

```
add_scan_groups g1 proc.g1
add_scan_chains c1 g1 scin scout
add_clocks 0 clk
set_system_mode analysis
report_nonscan_cells -tlatch
```

report_nonscan_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays the sequential non-scan model list.

Usage

```
report_nonscan_models [-Class { User | System}]
```

Description

Displays the sequential non-scan model list.

Displays sequential models that you added by using the [add_nonscan_instances](#) command.

Arguments

-Class {User | System}

An optional switch and literal pair that specifies the class code of the non-scan model that you specify. The valid literal names are as follows:

User — A literal that specifies that the list of non-scan models were previously added by using the [add_nonscan_instances](#) command. This is the default class.

System — A literal that specifies that the list of non-scan models were added by the tool.

Examples

The following example displays all sequential non-scan models from the non-scan model list:

```
set_system_mode analysis
add_nonscan_instances -modules d_flip_flop1 d_flip_flop2
report_nonscan_models
```

report_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays all the circuit points for which you do not want the tool to insert controllability and observability.

Usage

```
report_notest_points [-Paths] [-tool_identified]
```

Description

Displays all the circuit points for which you do not want the tool to insert controllability and observability.

The report_notest_points command displays the circuit points added using the [add_output_masks](#) command and which therefore, the tool cannot use for testability insertion.

You can also list the critical path definitions that added notest points by using the -Paths switch.

Arguments

- *>file_pathname*

An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- *>>file_pathname*

An optional redirection operator and pathname pair for appending to the contents of file_pathname.

- **-Paths (dft -test_points context only)**

An optional switch that displays the definitions for all currently loaded critical paths that have marked notest points. If this switch is not specified, the tool reports only the resulting notest points.

- **-tool_identified**

An optional switch that displays only the notest points added by the tool.

Note that if you issue the report_notest_points command without any switch, it will display the notest points added by the user.

Examples

The following example displays the list of all circuit points that the tool cannot use for testability insertion:

```
add_notest_points i_1006/o i_1007/o  
add_notest_points i_1009/o  
report_notest_points
```

Related Topics

[add_output_masks](#)

[delete_nottest_points](#)

report_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the currently masked primary output pins.

Usage

```
report_output_masks [[{> | >>} file_pathname] [-Display {FLAt_schematic}]]
```

Description

Displays a list of the currently masked primary output pins.

The report_output_masks command displays the masked primary output pins using the [add_output_masks](#) command. When you mask a primary output pin, the tool marks that pin as an invalid observation point during the fault detection process. The tool uses all unmasked primary output pins as possible observation points to which the effects of all faults propagate for detection.

Arguments

- `>file_pathname`
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- `>>file_pathname`
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.
- `-Display FLAt_schematic`
An optional switch and literal that specify to display the mask information for the primary output(s) as callouts in the Flat Schematic window.

Examples

The following example masks two primary outputs and then displays the results:

```
add_output_masks qb1 qb2
report_output_masks

TIEX qb1
TIEX qb2
```

Related Topics

- [add_output_masks](#)
- [delete_output_masks](#)
- [set_output_masks](#)

report_pattern_filtering

Context: dft -edt, patterns -scan

Mode: analysis

Displays the current pattern filtering configuration.

Usage

`report_pattern_filtering [<> | >>] file_pathname]`

Description

Displays the current pattern filtering configuration.

The report_pattern_filtering command displays the pattern filtering configuration set previously with the set_pattern_filtering command.

Note

 The report_environment command reports if pattern filtering is on, but shows no status for pattern filtering when it is turned off.

Arguments

- `>file_pathname`

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- `>>file_pathname`

An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the current pattern filtering configuration:

report_pattern_filtering

```
Current pattern filtering configuration
-----
original_patterns (internal) = 125
current_filtered_pattern(s) = 7
sampled_patterns_per_type = 2
sample_capture_clock = on
sample_pattern_type(s) = all
-----
```

Related Topics

[report_environment](#)

[write_patterns](#)

set_pattern_filtering

report_pattern_sets

Context: patterns ijttag

Mode: analysis

Creates a report of a specified pattern set or of all pattern sets.

Usage

```
report_pattern_sets [-sort_by {declaration_order | name}] [pattern_set_list]
```

Description

Creates a report of a specified pattern set or of all pattern sets.

This command creates a human-readable report of all pattern sets when the pattern_set_list argument is omitted. The report contains the names of the pattern sets, the associated timeplates, the number of vectors, the TAP start and TAP end states, and whether the pattern set has been saved.

Arguments

- `-sort_by {declaration_order | name}`

An optional switch and keyword pair that sorts the patterns in the report by order of declaration or by name. By default, the command sorts the patterns in the order of declaration. If you do not specify this switch, but do specify a pattern_set_list, the command reports pattern sets in the order specified by the list.

- `pattern_set_list`

An optional Tcl list of pattern set names. If this list is present, the report is restricted to the pattern sets in the list. Otherwise, the report contains all pattern sets.

Return Values

None

Examples

Example 1

The following example creates a report of all pattern sets sorted by name:

```
report_pattern_sets -sort_by name
```

Example 2

The following example creates a report of the pattern sets returned by the lsearch command:

```
report_pattern_sets [lsearch -inline -all [get_pattern_set_list] p*]
```

For a detailed example, refer to “[Create Pattern Sets](#)” in the *Tessent IJTAG User’s Manual*.

Related Topics

[close_pattern_set](#)
[get_open_pattern_set](#)
[get_pattern_set_list](#)
[open_pattern_set](#)
[reset_open_pattern_set](#)

report_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Displays information about patterns.

Usage

```
report_patterns [-EXTernal] [-PATtern_index pattern_index]  
    [-PATH_faults] [-CHain_test]  
    [-UNLoad_x]  
    [-INClude_always_pulsed_clocks]  
    [-X_Cell_report_limit { 20 | d | unlimited }]  
    [{>|>>} file_pathname]
```

Additional switches available only when EDT is on:

```
report_patterns  
    [-EDT [[-COnstant_chains] [-Verbose]]]
```

Description

Displays information about patterns.

You can display information for all patterns, or use the -Pattern_index switch to report on a single pattern. When you use the [set_pattern_filtering](#) command to create a temporary subset of sampled patterns, the report_patterns command reports only on patterns in the filtered subset. By default, information for internal patterns is reported, but you can report on patterns in an external pattern set by first specifying the external pattern file with the [read_patterns](#) command.

Also, by default, the report information will show pulse_always and pulse-in-capture clocks as an *. In that case, a note follows the report to identify the clock information.

The following information is reported for each pattern when the -EDT switch is not used:

- Pattern # — Pattern index
- Pre-filtering pattern # — Pattern index in the original unfiltered pattern set (reported only when filtering by the [set_pattern_filtering](#) command is in effect, causing “Pattern #” to show pattern indexes in the filtered subset).
- Pattern type — Basic, Clock PO, Clock sequential, MacroTest, Multi-load, RAM sequential or Iddq
- Cycles — An integer indicating how many capture cycles there are in the pattern
- Loads — An integer indicating how many scan loads there are in the pattern
- Observe procedure — Name of the observe procedure (Master_Observe or Shadow_Observe), if applied in the pattern

- Capture procedure — Name of the named capture procedure, if applied in the pattern
- Capture clock sequence — Sequence the clocks are pulsed in during capture

When you use the -EDT switch to report a pattern created with Tessent TestKompress (EDT On), the information content of the report changes to display the following EDT-specific information:

- EDT block name if applicable
- EDT channel index number
- EDT channel names
- EDT input channel pin names
- EDT output channel pin names
- Observed core scan chain name(s)

If a channel is masked in the reported pattern, the name of the single scan chain observed for that channel is listed. If the channel is not masked in the pattern, this means that all connected chains are observed and the “<all connected chains>” string displays.

Arguments

- **-EXTernal**
An optional switch that displays pattern information for external patterns instead of internal patterns. Before using this switch, you must specify the name of the external pattern file with the [read_patterns](#) command.
- **-PATtern_index *pattern_index***
An optional switch and integer pair that displays information only for the pattern with the specified index, not all patterns.
- **-PATH_faLts**
An optional switch that displays the path faults detected by the internal patterns. This switch cannot be used with the -external switch and is only valid when the current fault type is set to path_delay.
- **-CHain_test**
An optional switch that, together with -pattern_index switch, reports a particular chain test pattern in the pattern set. Without this switch, the tool reports scan test pattern information. Note that you can use this switch only when chain test patterns exist. Chain test patterns are created by the [write_patterns](#) command or if loaded as part of external patterns. If no chain test patterns exist, the tool issues an error message.
- **-EDT (Available only when EDT is on)**
An optional switch that (1) reports which of the internal scan chains connected to each scan channel are observed by the specified EDT pattern when used in conjunction with the

-Pattern_index switch, or (2) reports one line per pattern when used without the -Pattern_index switch. On each line, the pattern ID is followed by the pattern masking status for this pattern. This switch is valid only for Tesson TestKompress patterns created with Tesson TestKompress (EDT On).

- **-Constant_chains (Available only when EDT is on)**

An optional switch that together with the -EDT and -Pattern_index switches lists the actual data source (decompressor or constant 0) for the specified internal pattern. Use this switch to display test pattern information for the low-power decompressor.

- **-Verbose (Available only when EDT is on)**

An optional switch that together with the -EDT and -Pattern_index switches lists the actual names of all the connected chains in the observed_chains column. Without this switch, the summary message “<all connected chains>” is displayed rather than the names of all the connected chains.

- **-UNLoad_x**

An optional switch that reports patterns that contain some scan cells with X unload values. For EDT patterns, X values due to EDT masking are not considered as unload X. When you specify this switch with the -pattern_index switch, the command also reports the scan cells that capture X values (see “[Example 8](#)” on page 1706).

You can use the -unload_x switch for debugging purposes if you issue a “set_simulation_option -report_x_capturing_cells on” command and notice some patterns that capture X on scan cells. You should first use the -unload_x switch to see patterns that unload X, and then add the -pattern_index switch to further see which scan cells capture X of the specified pattern.

Use the -x_cell_report_limit switch to control the number of cells that get reported.

- **-INClude_always_pulsed_clock**

An optional switch that will enable the report to include the full names of the pulse-always and pulse-in-capture clocks. This changes the behavior from the default which is to represent these clocks as “*” in the report and follow the report with a note to identify the clocks.

- **-X_Cell_report_limit { 20 | d | unlimited }**

An optional switch that limits the number of capture X cells included in the report. You can limit the number of cells reported to a nonnegative integer (by default, 20), or you can specify no limit to the number of cells reported. If the report is truncated, a message indicates that the limit was reached and provides the total number of X captured cells from the report.

- **> file_pathname**

An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- `>>file.pathname`

An optional redirection operator and pathname pair for appending to the contents of `file.pathname`.

Examples

Example 1a

The following example of scan pattern reporting displays all patterns in the current internal pattern set. [Table 5-14](#) summarizes the display notation.

Table 5-14. Notation Used in report_patterns Display

Notation	Column	Meaning
master_obs	observe_proc	Pattern applies a master_observe procedure.
shadow_obs	observe_proc	Pattern applies a shadow_observe procedure.
-	observe_proc	No master_observe or shadow_observe procedure is applied.
-	capture_proc	No named capture procedure is applied.
-	capture_clock_sequence	No clock is pulsed in the capture window.
[-]	capture_clock_sequence	No clock is pulsed in the capture window, but primary inputs are being forced.
[clock1][clock2] ...	capture_clock_sequence	Sequence (left to right) in which <i>clock1</i> and <i>clock2</i> , etc. are pulsed.
[clock1, *]	capture_clock_sequence	By default, pulse-always and pulse-in-capture clocks are shown as *. A note is added at the end of the report to identify these clocks.
[clock1, clock2, ...]	capture_clock_sequence	Shows <i>clock1</i> and <i>clock2</i> , etc. are pulsed at the same time.
S	capture_clock_sequence	A scan load occurs in the corresponding cycle. ¹

1. The first cycle always has a scan load, so no S is displayed for the first cycle.

[report_patterns](#)

patt #	type	cycles	loads	observe_proc	cap_proc	cap_clk_seq
0	basic	1	1	master_obs	-	[clk1]
1	basic	1	1	-	-	[clk1]
2	clock_po	1	1	-	-	-
3	clock_seq	2	1	shadow_obs	named_cap1	[clk1] [clk1]
4	multi_load	3	3	-	-	[ramclk] S[clk1] S[clk2]
5	clock_seq	2	1	-	-	[clk1,clk2] [-]
6	clock_seq	2	1	-	-	[clk1] [-]
7	multi_load	3	2	-	-	[clk1] S[clk1] [clk1]
8	basic	1	1	-	-	[clk1]
9	basic	1	1	-	-	[-]
10	basic	1	1	-	-	[scn_rst]

Pattern 0 is a basic pattern that applies a master_observe procedure. Pattern 2 is a clock PO pattern; therefore, no clock is pulsed in the capture window. Pattern 3 is a clock sequential pattern that uses the named capture procedure “nmd_cap1” in which the clock sequence is [clk1] [clk1]. Pattern 4 is a multiple-load pattern. The S in the capture clock sequence indicates there is a scan load in the corresponding cycle (the first cycle always has a scan load, so no S is displayed there). Pattern 5 is a clock sequential pattern that pulses two clocks simultaneously in the first capture cycle, then in the cycle that follows, forces primary inputs without pulsing any clocks. Pattern 6 differs from pattern 5 in that it pulses just one clock in the first capture cycle.

Example 1b

Another example of report_patterns showing the case with pulse-always and pulse-in-capture clocks.

```
report_patterns

// pattern # type      cycles  loads   observe_proc  cap_proc  cap_clock_sequence
// -----
// 0    clock_sequential 2       1       -           -         [core_clk200,*] [core_clk200,*]
// 1    clock_sequential 2       1       -           -         [core_clk200,*] [core_clk200,*]
//
//Note: [*] = "int_shift_reg_clk1,int_shift_reg_clk2,int_shift_reg_clk3", which are pulse-
always clocks.
```

The “Note” line only appears if there are pulse-in-capture or pulse-always clocks in the design.

Example 2

The following example shows how you can use the [set_pattern_filtering](#) command to display a subset of patterns. The example first displays just the non-basic patterns of the preceding example, then displays just the patterns that use ramclk as a capture clock:

set_pattern_filtering -range 2 7

```
// Internal pattern filtering is on: 6 selected pattern(s), 11 original
// pattern(s).
```

report_patterns

		pre-filter		patt #	patt #	type	cycles	loads	observe_proc	cap_proc	cap_clk_s
0	2	clock_po	1	1	-	-	-	-	-	-	-
1	3	clock_seq	2	1	shadow_obs	named_cap1	[clk1]	[clk1]	[clk1]	S[clk1]	S[clk2]
2	4	multi_load	3	3	-	-	-	-	-	S[clk1]	S[clk2]
3	5	clock_seq	2	1	-	-	-	-	-	[clk1,clk2]	[-]
4	6	clock_seq	2	1	-	-	-	-	-	[clk1] [-]	[clk1]
5	7	multi_load	3	2	-	-	-	-	-	S[clk1]	[clk1]

set_pattern_filtering -clocks ramclk

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 1 selected pattern(s), 11 original
// pattern(s).
```

report_patterns

		pre-filter		patt #	patt #	type	cycles	loads	observe_proc	cap_proc	cap_clk_s	
0	4	multi_load	3	3	-	-	-	-	-	[ramclk]	S[clk1]	S[clk2]

Example 3

The following EDT pattern reporting example reports EDT chain test pattern 9. The design is a **Modular Compressed ATPG** design with two EDT blocks, p1 and p2. Block p1 has one channel and all chains connected to this channel are observed with this pattern. Block p2 has three channels. For each of them, only one chain is observed with this pattern.

report_patterns -pattern_index 9 -edt -chain_test

```
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// -----
// p1      1  p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1  <all conn.
//                                         chains>
// -----
// p2      1  p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1  core2_chain1
// "      2  p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2  core2_chain4
// "      3  p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3  core2_chain7
```

Example 4

The following example is the same as the preceding example except that the -Verbose switch is included in the command. This expands the “<all connected chains>” summary information to the names of all chains connected to p1_channel1.

```
report_patterns -pattern_index 9 -edt -chain_test -verbose
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// -----
// p1      1 p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1 core1_chain1
//                                core1_chain2
//                                core1_chain3
//                                core1_chain4
//                                core1_chain5
// -----
// p2      1 p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1 core2_chain1
// "      2 p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2 core2_chain4
// "      3 p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3 core2_chain7
```

Example 5

The following example reports on a particular pattern in an external pattern set for a design that has both **uncompressed scan chains** (direct access chains) and compressed scan chains. The direct access chains are reported in lines that begin “<uncompressed chain>”. As the design consists of just one block, the “block” column is unnecessary and it is not displayed.

```
read_patterns my_edt_patts
report_patterns -external -pattern_index 199 -edt -chain_test
// ch# channel_name channel_input_pin channel_output_pin obs_chains
// -----
// 1 pic_channel1 core1_edt_ch_in1  core1_edt_ch_out1  core1_chain1
// 2 pic_channel2 core1_edt_ch_in2  core1_edt_ch_out2  core1_chain4
// 3 pic_channel3 core1_edt_ch_in3  core1_edt_ch_out3  core1_chain7
// -----
// <uncompressed chain> core2_edt_ch_in1  core2_edt_ch_out1  core2_chain1
// <uncompressed chain> core2_edt_ch_in2  core2_edt_ch_out2  core2_chain2
```

Example 6

The following example sets the fault type to path_delay, loads path data into the system, and creates patterns to test the path delay faults. It uses the report_patterns command to check which path faults are detected by the internal patterns.

```
set_fault_type path_delay
read_fault_sites paths.txt
set_pattern_type -sequential 2
create_patterns
report_patterns -path_faults
```

```
// Pattern 0 detects following path fault(s):
//   Detected path = path1, edge = rise, detection = robust
//   Detected path = path44, edge = fall, detection = robust
// Pattern 1 detects following path fault(s):
//   Detected path = path3, edge = rise, detection = robust
//   Detected path = path12, edge = fall, detection = robust
//   Detected path = path16, edge = fall, detection = robust
//   Detected path = path21, edge = fall, detection = robust
//   Detected path = path33, edge = fall, detection = robust
//   Detected path = path34, edge = fall, detection = robust
//   Detected path = path45, edge = fall, detection = robust
//   Detected path = path52, edge = fall, detection = robust
// Pattern 2 detects following path fault(s):
//   Detected path = path5, edge = fall, detection = robust
//   Detected path = path44, edge = rise, detection = robust
...
...
```

Example 7

The following example shows the data source for scan chains associated with the 101 test pattern index.

```
report_patterns -edt -constant_chains -pattern_index 101
```

```

//          block           data source statistics
//  -----  -----
// piccpu1  8 of 8 scan chains are driven by constant value
//  -----
// piccpu2  3 of 8 scan chains are driven by constant value
//  -----
// piccpu3  8 of 8 scan chains are driven by constant value
//
// command: rep patterns -edt -constant_chains -patt 101 -verbose
//
//          block      chain      data source
//  -----  -----
// piccpu1 core1_chain1 constant 0
//          "       core1_chain2 constant 0
//          "       core1_chain3 constant 0
//          "       core1_chain4 constant 0
//          "       core1_chain5 constant 0
//          "       core1_chain6 constant 0
//          "       core1_chain7 constant 0
//          "       core1_chain8 constant 0
//  -----
// piccpu2 core2_chain1 decompressor
//          "       core2_chain2 decompressor
//          "       core2_chain3 constant 0
//          "       core2_chain4 constant 0
//          "       core2_chain5 decompressor
//          "       core2_chain6 constant 0
//          "       core2_chain7 decompressor
//          "       core2_chain8 decompressor
//  -----
// piccpu3 core3_chain1 constant 0
//          "       core3_chain2 constant 0
//          "       core3_chain3 constant 0
//          "       core3_chain4 constant 0
//          "       core3_chain5 constant 0
//          "       core3_chain6 constant 0
//          "       core3_chain7 constant 0
//          "       core3_chain8 constant 0
//

```

Example 8

The following example shows how to report patterns that contain some scan cells with X unload values:

report_patterns -unload_X

pattern #	type	cycles	loads	unload_X	observe_proc	capture_proc	capture_clock_seq
0	basic	1	1	yes	-	-	[clock]
1	basic	1	1	yes	-	-	[clock]
4	basic	1	1	yes	-	-	[clock]
5	basic	1	1	yes	-	-	[clock]
7	basic	1	1	yes	-	-	[clock]

You can then further investigate one of these patterns and report the scan cells that capture X values:

```
report_patterns -pattern_index 1 -unload_x
```

```
// pattern # type  cycles loads unload_X observe_proc capture_proc capture_clock_seq
// -----
//      0  basic    1      1     yes      -          -           [clock]
// Scan cell dff1 (233) captures X
// Scan cell dff3 (235) captures X
```

Example 9

The following example shows the default limit of 20 X captured cells reported:

```
report_patterns -unload_x -pattern_index 0
```

```
// pattern # type  cycles loads unload_x observe_proc capture_proc capture_clock_sequence
// -----
//      0  basic    1      1     Yes      -          cap1        [clk1,clk4]
//
// Scan cell '/reg_result_add_clk4_29/Q' (1274) captures X
// Scan cell '/reg_result_add_clk4_18/Q' (1262) captures X
...
// Scan cell '/reg_result_add_clk4_7/Q' (1170) captures X
// Only the first 20 out of 98 X captured cells are reported. Add "-X_cell_report_limit
// unlimited" to show all cells capturing X.
```

Related Topics

[read_cpf](#)

[read_upf](#)

[add_faults](#)

[delete_faults](#)

[read_faults](#)

[report_faults](#)

[set_power_metrics](#)

[add_primary_inputs](#)

[delete_primary_inputs](#)

[report_primary_outputs](#)

[add_primary_outputs](#)

[delete_primary_outputs](#)

[report_primary_inputs](#)

[report_capture_procedures](#)

[add_scan_groups](#)

[read_procfile](#)

[report_timeplate](#)

[write_patterns](#)

[write_procfile](#)

Multiprocessing to Reduce Runtime

[add_processors](#)

[delete_processors](#)

[add_read_controls](#)

[analyze_control_signals](#)

[delete_read_controls](#)

[add_register_value](#)

Serial Register Load and Unload for LogicBIST and ATPG

[set_lpct_controller](#)

[delete_register_value](#)

[set_register_value](#)

[set_pattern_buffer](#)

[report_processors](#)

report_power_data

Context: all contexts

Mode: all modes

Reports the power data currently loaded in the system from either a [read_cpf](#) or [read_upf](#) command.

Usage

```
report_power_data {[ -Summary | -All | -DOmains [domain_name ...] |  
    -SStates [state_name ...] } [-ISolation_cells] [-LLevel_shifters] [-Always_on_cells]  
    [-REtention_cells]
```

Description

Reports the power data currently loaded in the system from either a [read_cpf](#) or [read_upf](#) command.

Arguments

- **-Summary**

An optional switch that specifies the content of power data to be reported. By default, the command reports the summary of loaded power data.

- **-All**

An optional switch that reports the details of the loaded power data, including all power domains and all power modes. When executing the command in non-setup mode, the tool reports the count of low power cells associated with each power domain and the gate count of every power domains.

- **-DOmains [domain_name ...]**

An optional switch and optional repeatable string that reports only the specified power domains (*domain_name*) and the data associated with the specified power domains. If no power domain is specified, all power domains are reported.

- **-SStates [state_name ...]**

An optional switch and optional repeatable string that reports only the specified power mode and power data related to the specified power modes. If no power mode is specified, all power modes will be reported.

- **-ISolation_cells**

An optional switch that reports the detailed information of the isolation cells.

- **-LLevel_shifters**

An optional switch that reports the detailed information of the level shifters.

- **-Always_on_cells**

An optional switch that reports the detailed information of the always-on cells.

- **-REtention_cells**

An optional switch that reports the detailed information of the retention cells.

Examples

The following example displays detailed power data for isolation cells:

`report_power_data -isolation_cells`

Related Topics

[add_faults](#)

[delete_faults](#)

[read_faults](#)

[read_cpf](#)

[report_faults](#)

report_power_metrics

Context: dft -edt, patterns -scan

Mode: analysis

Displays shift and capture power metrics for specified test patterns.

Usage

```
report_power_metrics [-PATterns {ALL | pattern_index... | PAD
| RANGE first_pattern_index last_pattern_index}
[{-SORT_Capture [ST_Peak | ST_Average | WSA_Peak | WSA_Average]
|-SORT_Shift [LOAD | RESPonse]} [-ASCending | -DESCending]]
[-INTERNAL | -EXTernal] [-EVERY_CAPTURE_cycle]
[-INSTance instance_pathname...] [-MODULE module_name...]
[-SHIFT [-SCan_test | -CHain_test | -ALl_test]
-CYCLE {ALL | PEAK | cycle_index... | RANGE first_cycle_index last_cycle_index}]]]
[{> | >>} file_pathname]
```

Description

Displays shift and capture power metrics for specified test patterns.

Use this information to identify the test patterns that may cause chip failure due to power issues. Enter this command without any arguments to display a summary of shift and capture power metrics for all test patterns.

Note

 You can use this command when the [set_power_metrics](#) command is set to
-Shift off -Capture off and the test patterns are automatically resimulated to calculate shift and capture power if necessary.

Arguments

- **-PATterns {ALL | pattern_index... | PAD | RANGE *first_pattern_index* *last_pattern_index*...}**

An optional switch and literal, list of positive integers, or literal and list of positive integers that display the shift power and capture power for specified test patterns. Options include:

ALL — Displays the shift power and capture power for all patterns.

pattern_index — Displays the shift power and capture power for the specified patterns.
Pattern index starts from 0.

PAD — Displays the shift power consumed by the unloading of the last test pattern in the specified test patterns. This value is only available when [set_power_metrics -composite_shift_power](#) is enabled.

RANGE *first_pattern_index* *last_pattern_index* — Displays the shift power and capture power for all patterns from *first_pattern_index* to *last_pattern_index*.

- **-SORT_Capture [ST_Peak | ST_Average | WSA_Peak | WSA_Average]**

An optional switch and literal pair that sorts specified test patterns based on their switching activity during capture. Literal options include:

ST_Peak — Peak of state transitions. Default.

ST_Average — Average of state transitions.

WSA_Peak — Peak of weighted switching activity.

WSA_Average — Average of weighted switching activity.

For more information, on WSA and ST values, see Example 1 below and the [set_power_metrics](#) command.

- **-SORT_Shift [LOAD | RESPonse]**

An optional switch and literal pair that sorts the specified test patterns based on their function during test pattern application. This option is only valid when the -SHIFT option is specified. Literal options include:

LOAD — Switching activity during the loading of test pattern stimuli.

RESPonse — Switching activity during the unloading of test responses.

- **-ASCending | -DESCending**

An optional switch that sorts the specified test patterns in an ascending or descending order based on switching activity. By default, test patterns with the lowest switching activity are listed first (ascending order).

- **-INTernal | -EXTernal**

An optional switch that specifies whether power metrics are reported for the internal or external test pattern set. By default, the internal test pattern set is used.

- **-EVERY_CApture_cycle**

An optional switch that reports the weighted switching activity and the “state element transition” for every capture cycle on a per pattern basis. By default, the tool reports only the average and peak capture activity for the test pattern.

- **-INSTance *instance.pathname*...**

An optional switch and repeatable string that reports the shift and capture power usage for one or more specified design instances. The instance_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- **-MODULE *module.name*...**

An optional switch and repeatable string that reports the shift and capture power usage for all instances of a specified design module(s).

- [-SHIFT {-SCan_test | -CHain_test | -ALL_test}]

An optional pair of switches that reports shift and capture power metrics for a specific pattern type. By default, the power metrics for scan test patterns are reported. Options include:

-SCan_test— Scan test patterns. Default.

-CHain_test — Chain test patterns.

-ALL_test — Scan and chain test patterns.

- [-CYCLE {ALL | PEAK | *cycle_index* ... | RANGE *first_cycle_index* *last_cycle_index*}]

An optional switch and literal, list of integers, or a literal and a pair of integers that reports shift power for the specified test pattern cycles. You must enable the composite shift power calculation with [set_power_metrics](#) to calculate these values. Options include:

ALL — Shift transitions for all cycles.

PEAK — Shift transitions for the cycle with the peak shift power.

cycle_index — Shift transitions for the specified cycles.

RANGE *first_cycle_index* *last_cycle_index* — Shift transitions for a specified range of cycles.

- >*file_pathname*

An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >>*file_pathname*

An optional redirection operator and pathname pair for appending the contents of *file_pathname*.

Examples

Example 1

The following example displays the shift power and capture power for pattern 15.

```
report_power_metrics -patterns 15
```

Pattern	Shift			Capture		
	Load	Response	WSA	State	Element	Tran.
15	45.29%	40.56%	13.89% (15.49%)	14.32%	(18.13%)	

- The Shift Load number (45.29%) is the percentage of the number of scan cell transitions during the loading of the test stimuli over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.
- The Shift Response number (40.59%) is the percentage of the number of scan cell transitions during the unloading of the test response over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.

- The first Capture WSA (Weighted Switching Activity) number (13.89%) is the average of the weighted switching activity per capture cycle. The second number (15.49%) is the weighted switching activity for the peak capture cycle. The peak capture cycle is the cycle with the highest weighted switching activity among all capture cycles in a test pattern.

The WSA calculation includes the switching activity of the combinational logic, state elements, as well as factoring in gate output fanout for capture cycles.

- The first Capture State Element Tran. number (14.32%) is the average number of state element transitions per capture cycle. The second number (18.13%) is the number of state element transitions in the peak capture cycle. The peak capture cycle is the cycle with the highest number of state element transitions among all capture cycles in a test pattern.

Example 2

The following example displays a summary of capture power and shift power for all patterns:

report_power_metrics

Power Metrics	Min.	Average	Max.
WSA	9.03%	18.74%	36.82%
State Element Transitions	8.99%	25.44%	47.33%
Load Shift Transitions	39.06%	44.62%	51.66%
Response Shift Transitions	6.45%	39.38%	50.54%

Example 3

The following example calculates and reports the power metrics for external test patterns, *mypat.ascii*:

read_patterns mypat.ascii report_power_metrics -external

Power Metrics	Min.	Average	Max.
WSA	10.62%	12.54%	13.65%
State Element Transitions	13.59%	18.71%	20.94%
Peak Cycle			
WSA	10.62%	12.54%	13.65%
State Element Transitions	13.59%	18.71%	20.94%
Load Shift Transitions	15.38%	25.00%	30.77%
Response Shift Transitions	17.95%	21.15%	25.64%

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests for all test patterns in the internal test pattern set:

```
set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all

// Chain test :
//
//      Pattern      Shift
// Pat. loaded Pat. unloadedAverage
// -----
// 0      -          32.89%
// 1      0          48.60%
// 2      1          54.20%
// pad   2          49.56%
//
// Scan test :
//
//      Pattern      Shift
// Pat. loaded Pat. unloadedAverage
// -----
// 0      -          10.60%
// 1      0          8.60%
// 2      1          14.20%
// 3      2          8.20%
// pad   3          9.56%
```

Example 5

The following example shows the composite shift power for a specified range of cycles in the second and third test pattern in the internal test pattern set. In this example, the peak shift power for test pattern 1 occurs in cycle 6 as indicated by the asterisk cycle index:

```
set_power_metrics -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle range 4 6

//      Pattern      Cycle | Shift Transitions
// Pat. loaded Pat. unloaded
// -----
// 1      0          4      | 43.82%
//                   5      | 40.76%
//                   6      (*) | 47.64%
// 2      1          4      | 38.82%
//                   5      | 39.54%
//                   6      | 37.56%
//
// (*) indicates the cycle with the peak shift power.
```

Example 6

The following example reports the peak shift power for test pattern 1 and 2:

```
set power metric -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle peak
```

Pattern	Peak Cycle	Shift Transitions
// Pat. loaded Pat. unloaded		
// -----	-----	-----
// 1	0	5
// 2	1	114

Example 7

The following example shows the switching activity for capture when the -every_capture_cycle argument is specified. In this example, there are three capture cycles in pattern 10.

report power metrics -pattern 10 -every_capture_cycle

Pattern	Capture
//	WSA State Element Tran.
// -----	-----
// 10	8.23%, 25.21%, 24.68% 1.69%, 22.16%, 19.53%

The following example has patterns with different numbers of capture cycles. This occurs when NCPs or clock control definitions are used. This example shows patterns with 1, 2, and 3 capture cycles.

report power metrics -pattern range 12 15 -every_capture_cycle

Pattern	Capture
//	WSA State Element Tran.
// -----	-----
// 12	8.16%, 21.91%, 23.22% 1.54%, 16.64%, 19.08%
// 13	8.22%, 25.06%, 1.74%, 21.28%
// 14	8.17%, 21.47%, 23.40% 1.52%, 15.78%, 14.42%
// 15	8.21%, 1.61%,

Related Topics

[set_power_metrics](#)

report_primary_inputs

Context: all contexts

Mode: analysis, setup

Displays a list of the primary inputs of a circuit.

Usage

```
report_primary_inputs [-All | net.pathname... | primary_input_pin...]  
[-Class {Full | User | System}] [-Verbose] [{> | >>} file.pathname]
```

Description

Displays a list of the primary inputs of a circuit.

You can choose to display either the user class, system class, or full classes of primary inputs. Additionally, you can display all of the primary inputs or a specific list of primary inputs. If you issue the command without specifying any arguments, then the tool displays all the primary inputs.

Arguments

- **-All**
An optional switch that displays all the primary inputs. This is the default.
- **net.pathname**
An optional repeatable string that specifies the circuit connections whose user-class primary inputs to display.
- **primary_input_pin**
An optional repeatable string that specifies a list of system-class primary input pins to display.
- **-Class Full | User | System**
An optional switch and literal pair that specifies the source (or class) of the primary input pins to display. The literal choices are as follows:
 - Full** — Displays all the primary input pins in the user and system class. This is the default.
 - User** — Displays only the user-entered primary input pins.
 - System** — Displays only the netlist-described primary input pins.
- **-Verbose**
An optional switch that displays all the nets grouped under a primary input.
- **> file.pathname**
An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file.pathname*.

- *>> file_pathname*

An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the full classes of primary inputs:

```
add_primary_inputs indata2 indata4
report_primary_inputs -class full
```

Example 2

The following example displays the user classes of primary inputs:

```
add_primary_inputs /dut/*/CLK_X-pseudo_port_name MYCLKX
//Note: Adding primary input MYCLKX merging 224 nets.

report_primary_inputs -class user

USER: '/MYCLKX' (merged internal pin)
Merges 224 nets
```

Example 3

The following dft -scan example displays all of the primary inputs:

```
report_primary_inputs -all

SYSTEM:      '/clk'
SYSTEM:      '/datain'
```

The label “system” means that these are primary inputs that the tool automatically recognizes because they were in the netlist.

Related Topics

[add_primary_inputs](#)
[delete_primary_inputs](#)
[report_primary_outputs](#)

report_primary_outputs

Context: all contexts

Mode: analysis, setup

Displays the specified primary outputs.

Usage

```
report_primary_outputs [-All | net.pathname... | primary_output_pin...]  
[-Class {Full | User | System}] [{> | >>} file.pathname]
```

Description

Displays the specified primary outputs.

The report_primary_outputs command displays a list of primary outputs of a circuit. You can choose to display either the user class, system class, or full classes of primary outputs.

Additionally, you can display all of the primary outputs or a specific list of primary outputs. If you issue the command without specifying any arguments, then the tool displays all the primary outputs.

Arguments

- **-All**
An optional switch that displays all primary outputs. This is the default.
- **net.pathname**
An optional, repeatable string that specifies the circuit connections whose user-class primary outputs you want to display.
- **primary_output_pin**
An optional, repeatable string that specifies a list of system-class primary output pins that you want to display.
- **-Class Full | User | System**
An optional switch and literal pair that specifies the source (or class) of the primary output pins that you want to display. The literal choices are as follows:
 - Full** — A literal that displays all of the primary output pins in the user and system class. This is the default.
 - User** — A literal that displays only the user-entered primary output pins.
 - System** — A literal that displays only the netlist-described primary output pins.
- **> file.pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.

- *>> file.pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

Example 1

The following example displays all primary outputs in the user class:

```
add_primary_outputs outdata1 outdata3 outdata5
report_primary_outputs -class user
```

Example 2

The following dft -scan example displays all of the primary outputs:

```
report_primary_outputs -all
SYSTEM:      '/dataout'
SYSTEM:      '/dataout1'
```

The label “system” means that these are primary outputs that the tool automatically recognizes because they were in the netlist.

Related Topics

[add_primary_outputs](#)
[delete_primary_outputs](#)
[report_primary_inputs](#)

report_procedures

Context: dft -edt, dft -scan, patterns -scan, patterns -ijtag,
patterns -scan _retargeting, patterns -failure _mapping

Mode: setup, analysis

Displays the specified procedure.

Usage

```
report_procedures {procedure_name [group_name] [-Display Wave] | -All} [-COre  
    core_name [-MOde mode_name]]  
    [{>} | >>} file.pathname] [-EXpand_iprocs]
```

Description

Displays the specified procedure.

The report_procedures command displays all procedures or the specified procedure.

Arguments

- ***procedure_name***
A string that specifies which procedure to display.
- ***group_name***
An optional string that specifies a particular scan group from which to display the specified procedure.
- **-Display Wave**
An optional switch that displays the specified procedure in the Wave window of DFTVisualizer. This is for unoptimized NCPs. Use the [report_capture_procedures](#) command to display optimized NCPs.
- **-All**
A switch that specifies for the tool to display all procedures. This is the default.
- **-COre *core_name***
An optional switch and string pair that specifies that the tool reports the procedures for a given core. The core must exist, or an error will be issued.
- **-MOde *mode_name***
An optional switch and string pair that specifies that the tool reports the procedures for the given mode of a core defined by the -core *core_name* switch. The tool reports an error if this switch is specified without the -core switch. This switch is not necessary when there is only one mode for a core. If a core has more than one mode, an error will be issued if the mode is not specified.

- **>file.pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file.pathname.
- **>>file.pathname**
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file.pathname.
- **-EXPAND_iprocs**
An optional switch that reports the final solution that is used in the test_setup or test_end procedures for any embedded iCalls and iMerge in those procedures. If an iCall is used within an iMerge, then that entire iMerge is expanded. The start and end of the expanded cycles is marked by comments in the reported procedure or procedure file. Use this switch to when debugging E14 DRC violations.

Examples

Example 1

The following example displays all procedures:

report_procedures -all

Example 2

The following example shows using the -EXPAND_iprocs switch to work with the set of used ports and vector callbacks in:

ANALYSIS> report_procedure test_setup -expand_iprocs

```
procedure test_setup =
    timeplate tp1 ;
    iReset;
    // Begin expanded iCall m8051_B1_edt_i.setup edt_low_power_shift_en on ;
    // cycle 1 start at time 40
    cycle =
        annotate "+ Targets: ..."
        ...
        end;
    // End expanded iCall
    // cycle 5 starts at time 200
    cycle =
        ...
        end;
    end;
```

Example 2

This is an example of using -core and -mode.

report_procedures -core piccpu maxlen16_1 -mode

```
procedure capture my_cap =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        force core_1_edt_clock 0 ;
        force core_1_ramclk 0 ;
        pulse core_1_clk ;
    end;
end;
procedure capture =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        measure_po ;
        pulse_capture_clock ;
    end;
end;
procedure ram_sequential =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        pulse_read_clock ;
        pulse_write_clock ;
    end;
end;
procedure clock_sequential =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        pulse_capture_clock ;
        pulse_read_clock ;
        pulse_write_clock ;
    end;
end;
procedure shift =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        force core_1_edt_update 0 ;
        measure_sco ;
        pulse core_1_clk ;
        pulse core_1_edt_clock ;
    end;
end;
procedure load_unload =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force core_1_clk 0 ;
        force core_1_edt_clock 0 ;
        force core_1_edt_update 1 ;
        force core_1_ramclk 0 ;
        force core_1_reset 0 ;
```

```
        force core_1_scan_enable 1 ;
        pulse core_1_edt_clock ;
    end ;
    apply shift 16;
end;
procedure test_setup =
    timeplate gen_tp1 ;
// cycle 0 starts at time 0
cycle =
    force core_1_edt_clock 0 ;
    force core_1_reset 0 ;
    force core_1_scan_enable 0 ;
end;

end;
```

Example 3

This is an example of using -code with a specific procedure.

```
report_procedures load_unload -core piccpu maxlen16_1
```

```
procedure load_unload =
    timeplate gen_tp1 ;
// cycle 0 starts at time 0
cycle =
    force core_1_clk 0 ;
    force core_1_edt_clock 0 ;
    force core_1_edt_update 1 ;
    force core_1_ramclk 0 ;
    force core_1_reset 0 ;
    force core_1_scan_enable 1 ;
    pulse core_1_edt_clock ;
end ;
apply shift 16;

end;
```

Related Topics

[add_scan_groups](#)
[read_procfile](#)
[report_capture_procedures](#)
[report_timeplate](#)
[write_patterns](#)
[write_procfile](#)

report_processors

Context: unspecified, all contexts

Mode: setup, analysis, insertion (dft -edt context only)

Displays information about the processors used for the current multiprocessing environment.

Usage

```
report_processors [-Verbose]
```

Description

Displays information about the processors used for the current multiprocessing environment.

For more information, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

The report_processors command displays the information listed in [Table 5-15](#) for the current multiprocessing environment:

Table 5-15. Default report_processors Information Display

Column Name	Description
hosts	Network names of host machines
threads	The number of threads launched on that host
arch	Architecture of host machines (similar to entering “uname -m” on each host)
CPU(s)	Number of hardware processors and their speed for each host machine
%idle	Current CPU idle percentage (a small interval of time is sampled)
free RAM	A machine’s total free memory
process size	Size of the master or slave process

This information enables you to determine whether to use the current set of host machines for slave processes or whether to use different machines. For example, a host machine with a low %idle is not as likely to provide as much CPU time to slave processes as one that is 100% idle. The free RAM may also help you decide which machines to avoid at the present time.

Note

 Be sure to consider the number of processors on the host machine when evaluating the idle percentage (%idle), as a partially idle machine may have one or more completely idle processors available to run slave processes.

Arguments

- **-Verbose**

An optional switch that adds the data listed in [Table 5-16](#) to the normal display.

The master CPU time reported is cumulative. The slave CPU time reported is the sum of the CPU time for all the threads reported for the most recent distribution session.

Table 5-16. Additional Data Displayed by -Verbose

Column Name	Description
free swap	Total available swap space minus the swap space currently in use
total RAM	Total available RAM
total swap	Total available swap space
user CPU	Current sum of the time spent executing the master or slave process
sys CPU	Current sum of the time spent in system routines on behalf of the master or slave process

Examples

Example 1

The following example summarizes the performance statistics for the current multiprocessing environment:

report_processors

```
// hosts          threads   arch      CPU(s)    %idle    free
RAM   process size
// ----- -----
// nemo (master)    1 x86-64  24 x 2.8 GHz    6%     70229.83 MB  19320.18 MB
// ahab            4 x86-64  16 x 3.0 GHz   39%     15463.78 MB   458.61 MB
// master with 1 thread and 1 slave with 4 threads running.
```

Example 2

This example uses the **-verbose** switch for this command. The first “**report_processors -verbose**” command reports that the cumulative CPU time for the master prior to running **create_patterns** is 18m.

When you subtract the 18m CPU time reported by that command from the 11h 51m CPU time for the master reported by the second instance of the command, you have the total master CPU time for the **create_patterns** command, 11h 33m.

Please be aware that the CPU time reported for the slave threads is the sum of the thread CPU times for the latest distribution session.

The tabular output of this example is wrapped, as indicated with an ellipsis (...), for easy reading.

report_processors -verbose

```
// sub-command: report_processors -verbose
// hosts      threads      arch      CPU(s)      %idle      free RAM      ...
// -----  -----  -----  -----  -----  -----
// nemo (master)      1      x86-64    24 x 2.8 GHz      6%    70229.83 MB      ...
// ahab             4      x86-64    16 x 3.0 GHz     39%   15463.78 MB      ...

      free swap      total RAM      total swap      process size      user CPU      sys CPU
-----  -----  -----  -----  -----  -----
307199.70 MB  258086.16 MB  307199.99 MB  19320.18 MB  18m 5.3s  24.6s
47688.60 MB  129012.48 MB  47999.99 MB  458.61 MB   0.3s  0.1s
// master with 1 thread and 1 slave with 4 threads running.

// sub-command: create_patterns
...

// sub-command: report_processors -verbose
// hosts      threads      arch      CPU(s)      %idle      free RAM      ...
// -----  -----  -----  -----  -----  -----
// nemo (master)      1      x86-64    24 x 2.8 GHz     87%   97755.77 MB      ...
// ahab             4      x86-64    16 x 3.0 GHz     93%   35058.28 MB      ...

      free swap      total RAM      total swap      process size      user CPU      sys CPU
-----  -----  -----  -----  -----  -----
307199.70 MB  258086.16 MB  307199.99 MB  33636.34 MB  11h 51m 37s  2m 51.9s
47487.77 MB  129012.48 MB  47999.99 MB  30711.03 MB  18h 5m 6s  10m 8.1s
// master with 1 thread and 1 slave with 4 threads running.
```

Related Topics

[add_processors](#)[delete_processors](#)

report_read_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays all of the currently defined read control lines.

Usage

`report_read_controls [> | >>] file_pathname]`

Description

Displays all of the currently defined read control lines.

The report_read_controls command displays all the read control lines specified using the add_read_controls command. The display also includes the corresponding off-state with each read control line.

Arguments

- `> file_pathname`
An optional redirection operator and pathname pair for creating or replacing the contents of `file_pathname`.
- `>> file_pathname`
An optional redirection operator and pathname pair for appending to the contents of `file_pathname`.

Examples

The following example displays a list of the current read control lines:

```
add_read_controls 0 r1 r3
add_read_controls 1 r2 r4
report_read_controls
```

Related Topics

[add_read_controls](#)
[analyze_control_signals](#)
[delete_read_controls](#)

report_read_verilog_options

Context: unspecified, all contexts

Mode: all modes

Reports options specified with the [set_read_verilog_options](#) command.

Usage

```
report_read_verilog_options
```

Description

Reports options specified with the [set_read_verilog_options](#) command.

Arguments

None.

report_register_value

Context: all contexts

Mode: setup, analysis

Reports register information for each register name.

Usage

```
report_register_value [(> | >>) file_pathname]
```

Description

Reports register information for each register name.

In the report, the columns contain the following data:

- Register Value Name — Name of the register specified by the [add_register_value](#) command in the pattern generation dofile.
- Value string — Binary value loaded into the register, if known. The register is loaded when the [write_patterns](#) command is executed. Before [write_patterns](#) is executed, this column contain Xs for registers with dynamic contents. For register values with a width greater than 200, only the first 200 bits of the value are displayed.
- Width — Number of bits in the register.
- Dynamic/Static — Shows the DRC impact.
 - Dynamic: The contents of the register is not known/not used during DRC and can be changed.
 - Static: The contents of the register is known and used during DRC and cannot be changed.
- Source — The source of the register contents: user-defined or tool-determined.
- LSB/MSB First — Whether the register is applied from MSB > LSB or LSB > MSB.
- Pin Inversion — Indicates whether any inversions are present. Contains two values that refer to [input_pin_inversion](#) and [output_pin_inversion](#): F = false and T = true.

See “[Serial Register Load and Unload for LogicBIST and ATPG](#)” in the *Tessent Shell User’s Manual* for complete information.

Arguments

- $> file_pathname$

An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- *>> file.pathname*

An optional redirection operator and pathname pair for appending to the contents of *file.pathname*.

Examples

The following example reports the values of LPCT configuration data fields before and after the write_patterns command is executed. In this example, if you issue “[set_lpct_controller -max_shift 1000](#)”, the lpct_config_shift_length register has the width of 10, as shown below.

report_register_value

Report output before the write_patterns command is executed:

Register value name	Value string	Width	Dynamic/ Static	Source	LSB/ MSB First	Pin inversion
lpct_config_capture_depth	xx	2	dynamic	capture width	LSB first	FF
lpct_config_chain_pattern_load_count	XXXX XXXX XX	10	dynamic	chain patterns load/unload count	LSB first	FF
lpct_config_edt_bypass	0	1	static	user specified	MSB first	FF
lpct_config_reset_control	0	1	static	user specified	MSB first	FF
lpct_config_scan_en_control	0	1	static	user specified	MSB first	FF
lpct_config_scan_pattern_load_count	XXXX XXXX XXXX XXXX XXxx	20	dynamic	scan patterns load/unload count	LSB first	FF
lpct_config_shift_length	XXXX XXXX XX	10	dynamic	shift length	LSB first	FF

Report output after the write_patterns command is executed:

Register value name	Value string	Width	Dynamic / Static	Source	LSB/MSB First	Pin inversion
lpct_config_capture_depth	10	2	dynamic	capture width	LSB first	FF
lpct_config_chain_pattern_load_count	000001 0110	10	dynamic	chain patterns load/unload count	LSB first	FF
lpct_config_edt_bypass	0	1	static	user specified	MSB first	FF
lpct_config_reset_control	0	1	static	user specified	MSB first	FF
lpct_config_scan_en_control	0	1	static	user specified	MSB first	FF
lpct_config_scan_pattern_load_count	000000 000001 011111 11	20	dynamic	scan patterns load/unload count	LSB first	FF
lpct_config_shift_length	000010 1101	10	dynamic	shift length	LSB first	FF

Related Topics

- [add_register_value](#)
- [delete_register_value](#)
- [set_register_value](#)

[report_resources](#)

Context: unspecified, all contexts

Mode: all modes

Displays the machine resources used by the active tool session.

Usage

`report_resources`

Description

Displays the machine resources used by the active tool session.

The `report_resources` command displays the cumulative processing time and the process memory usage for the machine on which the tool is running.

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

Arguments

None

Examples

The following example reports on machine resources:

```
report_resources
// Machine Name: bongo
// User Name: smith
// User CPU Time: 27.8 seconds
// System CPU Time: 1.6 seconds
// Memory Used: 141.739 MB
```

Related Topics

[set_pattern_buffer](#)

[report_processors](#)

report rtl_to_gates_mapping

Context: all contexts

Mode: all modes

Reports the name-mapping rules for RTL to post-synthesis/post-layout name mapping.

Usage

```
report rtl_to_gates_mapping
```

Description

Reports the name-mapping rules for RTL to post-synthesis/post-layout name mapping specified that exists by default or was added using the [add rtl_to_gates_mapping](#) command.

See the [add rtl_to_gates_mapping](#) command description to know what those mappings are for, and how to customize them.

Arguments

None.

Examples

The following example calls the report_rtl_to_gates_mapping before any RTL to gates mappings were added using the [add rtl_to_gates_mapping](#) command. In this case, the report shows the built-in default mappings. The [add rtl_to_gates_mapping](#) command is used to add an instance name mapping. The second report_rtl_to_gates_mapping invocation shows the added mapping.

report rtl_to_gates_mapping

```
// Index Register name map
// -----
// 1   '\% (name)_reg '
// 2   '\% (name)_reg_%(d0) '
// 3   '\% (name)_reg[%(d0)] '
// 4   '\% (name)_reg[%(d0)][%(d1)] '
// 5   '\% (name)_reg[%(d0)][%(d1)][%(d2)] '
// 6   '\% (name)_reg[%(d0)][%(d1)][%(d2)][%(d3)] '
// 7   '% (name)_reg_%(d0) '
// 8   '% (name)_reg_%(d0) '
// 9   '% (name)_reg_%(d0)_%(d1) '
// 10  '% (name)_reg_%(d0)_%(d1)_%(d2) '
// 11  'rtlcreg_%(name)_%(d0)' 
```

```
add rtl_to_gates_mapping -instance_name_map_list {corea/coreb corea}
report rtl_to_gates_mapping
```

```
// Index Register name map
// -----
// 1  '\% (name)_reg '
// 2  '\% (name)_reg_%(d0) '
// 3  '\% (name)_reg[%(d0)] '
// 4  '\% (name)_reg[%(d0)][%(d1)] '
// 5  '\% (name)_reg[%(d0)][%(d1)][%(d2)] '
// 6  '\% (name)_reg[%(d0)][%(d1)][%(d2)][%(d3)] '
// 7  '%(name)_reg_%(d0)'
// 8  '%(name)_reg_%(d0)_'
// 9  '%(name)_reg_%(d0)_%(d1)_'
// 10 '%(name)_reg_%(d0)_%(d1)_%(d2)_'
// 11 'rtlcreg_%(name)_%(d0)'

// Index Original instance name Mapped instance name
// -----
// 1      corea/coreb          corea
```

Related Topics

[add_rtl_to_gates_mapping](#)

report_run_synthesis_options

Context: unspecified, dft, patterns

Mode: setup, analysis, insertion

Reports the values for all options set with the [set_run_synthesis_options](#) command.

Usage

```
report_run_synthesis_options
```

Description

Reports the values for all options set with the [set_run_synthesis_options](#) command.

Arguments

None.

Examples

The following example demonstrates using of this command:

```
SETUP> set_run_synthesis_options dc_shell -startup_file .synopsys_dc.setup
SETUP> report_run_synthesis_options

// Synthesis options
// -----
// Option           Value
// -----
// default_synthesis_tool    dc_shell
// synthesis_output_directory .synthesis_outdir
// compilation_options      {}
// pre_compilation_commands {set_scan_configuration -style multiplexed_flip_flop}
// post_compilation_commands {change_names -rules verilog -hier}
// startup_file          .synopsys_dc.setup
// command_name          dc_shell
//
```

Related Topics

[check_synthesis](#)
[get_run_synthesis_options](#)
[run_synthesis](#)
[set_run_synthesis_options](#)

report_scan_cells

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: analysis, insertion (dft -scan context only)

Displays a report on the scan cells that reside in the specified scan chains.

Usage

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

```
report_scan_cells [-All | {chain_name [{-Range cell_id1 cell_id2} | cell_id...]}...]  
[-VERilog] [-Power_domain] [{> | >>} file_pathname]
```

Context: dft -scan

```
report_scan_cells [-All | chain_name...] [-SHift_register_flops]  
[-Power_domain] [-SCan_mode scan_mode] [-Filename filename [-Replace]]  
[{> | >>} file_pathname]
```

Context: dft -test_points -no_rtl

```
report_scan_cells [-All | chain_name...] [-SHift_register_flops]  
[-Power_domain] [-Filename filename [-Replace]] [{> | >>} file_pathname]
```

Description

Displays a report on the scan cells that reside in the specified scan chains.

The report_scan_cells command provides a report on the scan cells within specific scan chains. The report provides the following information for each scan cell:

- Chain cell index number (cell ID), where 0 is the scan cell closest to the scan-out pin.
- Scan chain in which the scan cell resides.
- Scan group in which the scan cell resides. (The scan group data will be hidden if the design has only scan group.)
- Sequential element type, primitive type, and latch triggering type during shifting. The latch triggering type can be one of the following: LE, TE, AH, AL.
- Inversion data for each sequential element.
- Gate index number for each sequential element.
- Shift clock name and inversion status at the clock input of each sequential element. (“dft -edt” and “patterns -scan” only)
- Top-level library model cell name of each sequential element.
- Instance name of each sequential element.
- Library instance name, if one is defined.

- Cell input and output pins.

For more information, see “[Scan Terminology](#)” in the *Tessent Scan and ATPG User’s Manual*.

If you issue the command without specifying any arguments, then the tool displays a report on the scan cells for all scan chains.

Tip

 If you are saving a log of the tool session, reporting all scan cells will fill the logfile with information for every scan cell—a lot of information if the design is large! To reduce the impact on the logfile in this case, consider reporting on only one scan chain.

Note

 Although scan cells are listed in the order of nearest-to-output first, the sequential elements inside one cell are not listed in that order (the master is always listed first).

Be aware that when reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool reads the cell name, “FFH\E\MY_SCAN_FF .SCAN_FF5” in a WGL pattern, as “FFH/E/MY_SCAN_FF/SCAN_FF5”, storing the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note

 The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

dft -scan only: If you issue the command with the -Filename switch, then the tool writes the scan cells to a file in the format that can be read by the [insert_test_logic](#) command. The format of the written file is different than the format of the viewed report. You can optionally edit the scan cell order in the file before reading the file with the [insert_test_logic](#) command.

Arguments

- -All

An optional switch that displays information for all the scan cells in all the scan chains. This is the default.

- chain_name

An optional, repeatable string that specifies a scan chain whose scan cell data you want to display. When you use this argument without accompanying cell IDs, the tool displays information for all the scan cells in the scan chain. If you follow the chain_name argument with the -Range switch (and beginning and ending cell IDs for the range), the tool displays information for every cell in the range. If you provide one or more cell IDs without the -Range switch, the tool displays information for just those cells in that chain.

- -Range *cell_id1* *cell_id2*
An optional switch and pair of integers that specify the beginning and ending cell IDs of a range of scan cells in *chain_name* whose data you want to display.
- *cell_id*
An optional repeatable integer that specifies the index number (cell ID) of a scan cell and, together with the *chain_name* argument, identifies a specific scan cell whose data you want to display.
- -VERilog
An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.
- -SHift_register_flops (dft -scan only)
An optional switch that specifies to print only the shift register flip-flops that are stitched into scan chains.
- -Filename *filename* [-Replace] (dft -scan only)
An optional switch and string that specifies that the tool writes the list of scan cells to a file. The format of the written file is different than the format of the viewed report; see the [insert_test_logic](#) filename argument for detailed information on the file format. The *-Replace* switch specifies that the file should be overwritten if it already exists.
- -Power_domain
An optional switch that creates a “Power Domain” column in the report when applicable.
- -Scan_mode *scan_mode*
An optional switch and string pair that specifies the scan mode for which you want to tool to report scan cell information.
- >*file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- >>*file_pathname*
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

This example displays a list of the scan cells for scan chain0. The design only has one scan group; therefore, the scan group column is not shown.

```
report_scan_cells chain0
```

cell#	chain	memory_type	inv	gate#	shift_clock	inv	cell_name	instance_name	(ext.pin names)
0	chain0	MASTER (LA-AH)	FFTT	24848	/clk	F	DFF3	/Q_I4	" "
1	chain0	MASTER (LA-AH)	FFFF	25040	/clk	F	DFF1	/H_I4	" "
		SLAVE (FF-TE)	FFFF	24849	/clk	T	DFF2	/Q_I4	" "
2	chain0	MASTER (LA-AH)	FFFF	24546	/clk	F	DFF1	/R_I1	" "
		SLAVE (FF-TE)	FFFF	25041	/clk	T	DFF2	/Q_I4	" "
3	chain0	MASTER (LA-AH)	FFFF	24542	/clk	F	DFF1	/R_I1	" "
		SLAVE (FF-TE)	FFFF	24547	/clk	T	DFF2	/Q_I4	" "
4	chain0	MASTER (LA-AH)	FFFF	24914	/clk	F	DFF1	/S_I1	" "
		SLAVE (FF-TE)	FFFF	24543	/clk	T	DFF2	/Q_I4	" "
5	chain0	MASTER (LA-AH)	FFFF	24544	/clk	F	DFF1	/Z_I4	" "
		SLAVE (FF-TE)	FFFF	24915	/clk	T	DFF2	/Q_I4	" "
6	chain0	MASTER (LA-AH)	FFFF	24540	/clk	F	DFF2	/N_I3	" "
		SLAVE (FF-TE)	FFFF	24545	/clk	T	DFF2	/Q_I4	" "

The first column in the report displays the chain cell index number, where 0 is the scan cell closest to the scan-out pin.

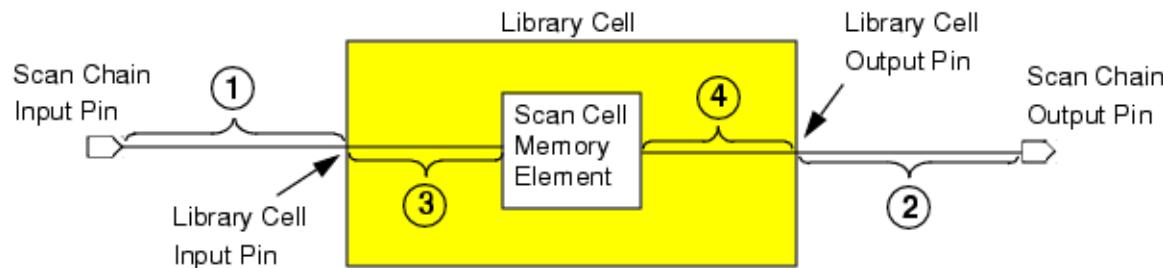
The second column displays the name of the scan chain the scan cell is in.

The third column displays the scan cell's sequential element type(s), primitive type, and latch triggering type during shifting. The latch triggering type can be any of the following: LE, TE, AH, AL.

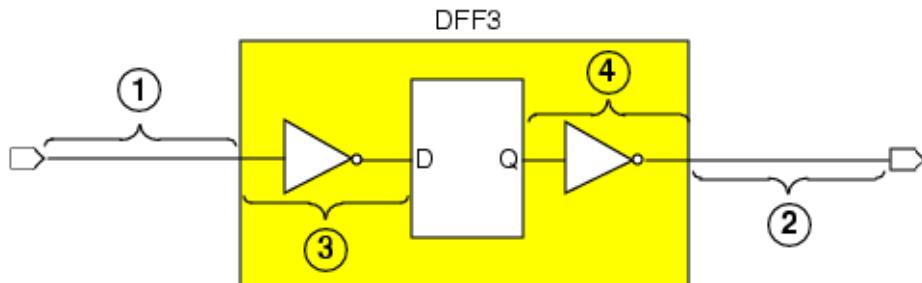
The fourth column (inv) contains inversion data using F (false) to indicate no inversion difference and T (true) to indicate inversion difference. The inversion data has four parts for each scan cell memory element. The report presents these details (left-to-right) as follows:

- Inversion of the signal between the scan chain input pin and the library cell input pin.
- Inversion of the signal between the library cell output pin and the scan chain output pin.
- Inversion of the signal between the library cell input pin and the scan cell memory element (primitive).
- Inversion of the signal between the scan cell memory element (primitive) and the library cell output pin.

The following diagram illustrates the four parts of the “inv” column:



For example, scan cell 0 in the previous scan cell report for chain0 shows inversion data FFTT. This indicates that library cell DFF3 contains an inverter on the input data path as well as one on the output path as shown here:



The inversion data is useful when comparing scan cell data reported in the tool (“set_gate_report -pattern_index n”) to values stored in the pattern file. Since different pattern formats handle inversions differently, the scan cell inversion data can also be useful in understanding differences between pattern formats.

The fifth column displays the gate index number.

The sixth column displays the name of the shift clock for the scan cell.

Note

 In the rare case where a single scan cell’s clock port is driven by multiple shift clocks at the same time, the tool reports the data for only one of the clocks. An example would be two different shift clocks with identical waveforms ANDed together to drive the scan cell’s clock port (not recommended due to the potential for clock skew).

The seventh column shows whether the shift clock’s defined inactive state corresponds to the cell’s clock input value that for level-sensitive devices results in the cell becoming inactive, or for edge-triggered devices is the initial value of a capturing transition. If they correspond (no inversion), an F is listed. If they do not correspond (inversion), a T is listed. For more information on the meaning of “inactive” as it applies to scan clocks, refer to the [add_clocks](#) command description.

Note

 If the scan cell is a hierarchical model, inversion is indicated with respect to the clock ports of sequential element primitives instantiated within the model, not with respect to the clock input(s) at the top level of the model.

The eighth column displays the cell name of the sequential element.

The ninth column displays the instance name of the sequential element.

The tenth column displays the library instance name. If there is no library instance name, the report shows two double-quotes in the library instance name field.

Note

To see the name of the library model corresponding to a particular scan cell, open the Design Browser window of DFTVisualizer with the [open_visualizer](#) command. The Design Browser will open by default, and with the Instance tab active in the Design Browser, find the instance name of the scan cell. The library model is listed to the right of the instance name.

The last column displays the library cell input and output pins of the scan cell. If the scan cell input or output pin does not directly connect to the library cell boundary pin, the report shows a dash (-) in the corresponding pin field.

Example 2

The following example displays a list of all scan cells in the analysis system mode:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_system_mode analysis
report_scan_cells
```

CellNo	Chain Name	Group Name	Pathname	CellName	ScanOut	Clock	Clock Polarity
0	chain1	group1	/MQ_I400	sffr	Q	clk2	(+)
1	chain1	group1	/FH_I400	sffr	QB	clk2	(+)
2	chain1	group1	/FQ_I10	sffr	QB	clk2	(+)
-	chain1	group1	/lckup1	latch	Q	clk1	(-)
3	chain1	group1	/RP_I10	sffr	Q	clk1	(+)
4	chain1	group1	/IS_I10	sffr	Q	clk1	(+)
5	chain1	group1	/CZ_I400	sffr	QB	clk1	(+)

- The first column displays the chain cell index number, where 0-0 is the scan cell closest to the scan-out pin.
- The second column displays the chain name where the scan cell resides.
- The third column displays the group name where the scan cell resides.
- The fourth column displays the hierarchical path of the scan cell.
- The fifth column displays the library model name for the scan cell.
- The sixth column displays the scan out port of the scan cell.
- The seventh column displays the clock for the scan cell.
- The eighth column displays the polarity of the clock of the scan cell.

Example 3

The following dft -scan example adds the new column ShiftRegID/CellNo to the report when it identifies a shift register in the netlist. This column contains a tool-assigned number for the shift register ID and a cell number that indicates the order in which the flip-flops are originally connected in the shift register structures. The column contains “-/-” for those cells that are not part of a shift register.

report_scan_cells

CellNo	Name	Polarity	Chain	Group	ShiftReg		Clock	
			Name	Pathname	ID/CellNo	CellName		
0	chain1	dummy	/ud5	-/-	sff	Q	clk	(+)
1	chain1	dummy	/ud6	-/-	sff	Q	clk	(+)
2	chain1	dummy	/ud4	1/4	dff	Q	clk	(+)
3	chain1	dummy	/ud3	1/3	dff	Q	clk	(+)
4	chain1	dummy	/ud2	1/2	dff	Q	clk	(+)
5	chain1	dummy	/ud1	1/1	sff	QB	clk	(+)

Example 4

The following dft -scan example uses the -Shift_register_flops switch to print only the shift register flops in the report:

report_scan_cells -shift_register_flops

CellNo	Name	Polarity	Chain	Group	ShiftReg		Clock	
			Name	Pathname	ID/CellNo	CellName		
2	chain1	dummy	/ud4	1/4	dff	Q	clk	(+)
3	chain1	dummy	/ud3	1/3	dff	Q	clk	(+)
4	chain1	dummy	/ud2	1/2	dff	Q	clk	(+)
5	chain1	dummy	/ud1	1/1	sff	QB	clk	(+)

Example 5

The following example shows use of the -Power_domain switch to report on power domains.

report_scan_cells -power_domain

CellNo	ChainName	GroupName	Pathname	CellName	ScanOut	Clock	ClockPolarity	PowerDomain
0	chain1	dummy	/I2/df_0001_1	sff	QB	clk	(+)	PD2
1	chain1	dummy	/I2/df_0002_1	sff	QB	clk	(+)	PD2
2	chain1	dummy	/I2/df_0003_1	sff	QB	clk	(+)	PD2
0	chain2	dummy	/I10/df_0003_1	sff	Q	clk	(+)	PD3
1	chain2	dummy	/I10/df_0001_1	sff	QB	clk	(+)	PD3
2	chain2	dummy	/I10/df_0002	sff	QB	clk	(+)	PD3
0	chain3	dummy	/I6	sff	QB	clk	(+)	PD1

Example 6

The following dft -scan example shows the additional output that is reported when sub-chains are encountered. Specifically, the starting and ending cell in a sub-chain are listed as a range value in the CellNo column, the clock for the first and last cell in the sub-chain are listed in the Clock column, and the polarity of each clock is listed in the Clock Polarity column.

CellNo	Chain	Group				Clock	
CellNo	Name	Name	Pathname	CellName	ScanOut	Clock	Polarity
0	chain1	grp1	/usf2	SCIFTD11S10	SO	sysCLK	(+)
1	chain1	grp1	/usf1	SCIFTD11S10	SO	sysCLK	(+)
0-3	chain2	grp1	/um1	&subchain1	SO	sysCLK,sysCLK	(+,+)
4-7	chain2	grp1	/uB1/um1	&subchain1	SO	sysCLK,sysCLK	(+,+)
8-9	chain2	grp1	/uC1	&subchain2	so	sysCLK,sysCLK	(+,+)
10	chain2	grp1	/uff2	SCIFTD11S10	SO	sysCLK	(+)
11	chain2	grp1	/uB1/uff2	SCIFTD11S10	SO	sysCLK	(+)
0-3	chain3	grp1	/um2	&subchain1	SO	MYTCLK,MYTCLK	(+,+)
4-7	chain3	grp1	/uB2/um2	&subchain1	SO	MYTCLK,MYTCLK	(+,+)
8-11	chain3	grp1	/uB2/um1	&subchain1	SO	MYTCLK,MYTCLK	(+,+)
12-15	chain3	grp1	/uB1/um2	subchain1	SO	MYTCLK,MYTCLK	(+,+)
16-17	chain3	grp1	/uC2	&subchain2	so	MYTCLK,MYTCLK	(+,+)
18	chain3	grp1	/uff1	SCIFTD11S10	SO	MYTCLK	(+)
19	chain3	grp1	/uB1/uff1	SCIFTD11S10	SO	MYTCLK	(+)
20	chain3	grp1	/uB2/uff1	SCIFTD11S10	SO	MYTCLK	(+)
21	chain3	grp1	/uB2/uff2	SCIFTD11S10	SO	MYTCLK	(+)

Related Topics

[add_scan_chains](#)

[add_scan_groups](#)

report_scan_chains

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all

Displays a report on all the current scan chains.

Usage

patterns -scan Usage

```
report_scan_chains [-All_blocks] [-SUBchains [-Verbose]] [-Inversion]  
[-INVALID_scan_pins] [{> | >>} file_pathname]
```

dft -scan Usage

```
report_scan_chains [{> | >>} file_pathname]
```

Description

Displays a report on all the current scan chains.

The report shows the following information for each scan chain:

- Name of the scan chain
- Name of the scan chain group
- Scan chain input and output pins
- Length of the scan chain
- Name of the scan clock(s) (but only after you enable the E8 DRC in setup mode)

If the DRC E8 handling is set to anything other than Ignore when leaving Setup system mode, the command additionally lists the clocks specified to be pulsed in the shift procedure in the test procedure file. This enhanced report is only available in non-Setup modes. The clocks reported are likely a superset of the clocks that are actually used to clock the scan cells. For a precise report of the shift clocks, use the [report_scan_cells](#) command.

Arguments

- `-All_blocks`

Note



This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks.

An optional switch that displays information for all scan chains in every modular EDT block. Without this switch, only information defined for the current EDT block displays. If a current EDT block is not defined, the information for all EDT blocks displays. Use the current EDT block using either the [add_edt_blocks](#) or [set_current_edt_block](#) command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it

controls/observes. For more information, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

When using the TCD-based automated flow, the tools adds all EDT blocks and does not set the current block. To report the scan chains in this case, you must use the `-all_blocks` switch or use `set_current_edt_block` to identify the blocks.

- **-SUBchains**

An optional switch that displays information about the subchains of the scan chains, including the number of required serial shifts and the number of scan cells in each subchain. Only the scan chains with a subchain display.

- **-Verbose**

An optional switch that displays detailed information about the scan cells in each subchain.

- **-Inversion**

An optional switch that displays information about whether there is any net inversion across the entire length of the scan chain, between the scan input pin and scan output pin. For EDT only, this switch also displays whether there is any inversion between the decompressor output and the input of the scan chain and whether there is any inversion between the scan chain output and the compactor output. This switch can be used in all modes except Setup mode.

- **-INVALID_scan_pins**

An optional switch that displays only scan chains that have invalid scan pin definitions (input pin and/or output pin). When using this option, in addition to filtering the chains, the tools also adds “(invalid)” next to every non-existent pin. See “[Example 6](#)” on page 1749.

In some cases when EDT Finder is on, you may only end up with invalid pins because the tool relaxes the pin checks when a chain is added. Having any invalid pins means EDT Finder cannot be turned off; this switch allows you to find invalid pins that prevent EDT Finder from being disabled.

- **> file_pathname**

An optional redirection operator and pathname pair for creating or replacing the contents of `file_pathname`.

- **>> file_pathname**

An optional redirection operator and pathname pair for appending to the contents of `file_pathname`.

Examples

Example 1

The following example displays a report of all the scan chains and their scan clocks. Note that because the tool ignores the E8 DRC by default, the command does not display scan clock information.

ATPG> report_scan_chains

```
chain = chain1 group = grp1 input = /scan_in1 output = /scan_out1 length = 27
chain = chain2 group = grp1 input = /scan_in2 output = /scan_out2 length = 28
chain = chain3 group = grp1 input = /scan_in3 output = /scan_out3 length = 28
```

However, when you enable the E8 DRC in setup mode, the tool stores and reports scan clock information:

```
SETUP> set_drc_handling e8 note
SETUP> set_syststem_mode atpg
ANALYSIS> report_scan_chains
chain = chain1 group = grp1 input = /scan_in1 output = /scan_out1 length = 27
master_clock(s) = /SCK1
chain = chain2 group = grp1 input = /scan_in2 output = /scan_out2 length = 28
master_clock(s) = /SCK2
chain = chain3 group = grp1 input = /scan_in3 output = /scan_out3 length = 28
master_clock(s) = /SCK3
```

Example 2

The following example displays a report of all the subchains.

```
report_scan_chains -subchains
chain = c1 group = g1 input = /SI0 output = /SO0 length = 11
  subchain-1 from chain "c1" cell 0 to cell 4, length = 5
    1 sub-chain in chain "c1", maximum subchain length within the chain is 5
chain = c3 group = g1 input = /SI2 output = /SO2 length = 7
  subchain-1 from chain "c3" cell 1 to cell 5, length = 5
    1 sub-chain in chain "c3", maximum subchain length within the chain is 5

The following 2 library models include sub-chains:
library_name max_subchain_length
----- -----
macro_sff1           5
reg5                 5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel
shifts.
```

Example 3

The following example indicates there is net inversion across the entire length of the scan chain, there is no inversion between the decompressor output and the input of the scan chain, and there is inversion between the scan chain output and the compactor output.

```
report_scan_chains -inversion
chain = chain1 group = grp1 input = /cpu_i/scan_in1 output =
/cpu_i/scan_out1 length = 152 inversion = Yes decompressor_inversion = No
compactor_inversion = Yes
```

Example 4

The following example shows reporting when power domains are present in the design:

report_scan_chains

```
Power domain 'PD2' chains:  
-----  
chain = chain1 group = dummy input = /scan_in1 output = /scan_out1  
length = 3 scan_enable = /scan_en clock = /clk  
  
Power domain 'PD3' chains:  
-----  
chain = chain2 group = dummy input = /scan_in2 output = /so2 length = 3  
scan_enable = /scan_en clock = /clk  
  
Power domain 'PD1' chains:  
-----  
chain = chain3 group = dummy input = /scan_in3 output = /scan_out2  
length = 1 scan_enable = /scan_en clock = /clk
```

Example 5

The following example displays a detailed report of the scan cells in the subchains.

report_scan_chains -sub -verbose

```
chain = c1 group = g1 input = /SI0 output = /SO0 length = 11  
    sub-chain-1 from chain "c1" cell 0 to cell 4, length = 5  
-----  
cell# chain memory_type gate# library_name instance_name (ext.pinnames)  
-----  
0    c1    MASTER (FF-LE) 147 macro_sff1 /macro1  ""  (-,Q)  
1    c1    MASTER (FF-LE) 148 macro_sff1 /macro1  ""  (-,-)  
2    c1    MASTER (FF-LE) 149 macro_sff1 /macro1  ""  (-,-)  
3    c1    MASTER (FF-LE) 150 macro_sff1 /macro1  ""  (-,-)  
4    c1    MASTER (FF-LE) 151 macro_sff1 /macro1  ""  (SI,-)  
1 sub-chain in chain "c1", maximum sub-chain length within the chain is 5  
chain = c3 group = g1 input = /SI2 output = /SO2 length = 7  
    sub-chain-1 from chain "c3" cell 1 to cell 5, length = 5  
-----  
cell# chain memory_type gate# library_name instance_name (ext.pinnames)  
-----  
1    c3    MASTER (FF-LE) 164 reg5      /macro5/reg0 s_5   (-,SO)  
2    c3    MASTER (FF-LE) 165 reg5      /macro5/reg0 s_4   (-,-)  
3    c3    MASTER (FF-LE) 166 reg5      /macro5/reg0 s_3   (-,-)  
4    c3    MASTER (FF-LE) 167 reg5      /macro5/reg0 s_2   (-,-)  
5    c3    MASTER (FF-LE) 168 reg5      /macro5/reg0 x_1   (SI,-)  
1 sub-chain in chain "c3", maximum sub-chain length within the chain is 5
```

The following 2 library models include sub-chains:

library_name	max_subchain_length
macro_sff1	5
reg5	5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel shifts.

Example 6

In the following example, the -INValid_scan_pins switch is used to display scan chains that have invalid scan pin definitions:

```
SETUP> report_scan_chains -all_blocks -invalid_scan_pins

EDT block = 'm1_28x16' chain = m1_chain5 group = grp1 input =
  'invalid_scanin1' (invalid) output = 'invalid_scanout1' (invalid) length
  = unknownEDT block = 'm2_4x32' chain = m2_chain1 group = grp1
  input = 'invalid_scanin2' (invalid)
  output = '/core2_4x32/retimetest_i/edt_so1' length = unknownEDT
  block = 'm2_4x32' chain = m2_chain2 group = grp1
  input = 'invalid_scanin3' (invalid)
  output = '/core2_4x32/retimetest_i/edt_so2' length = unknown
```

Related Topics

[add_scan_chains](#)
[delete_scan_chains](#)
[report_scan_cells](#)
[set_parallel_load_subchains](#)

report_scan_elements

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all modes

Reports the scan elements that are created by the tool for scan insertion purposes.

Usage

```
report_scan_elements [object_spec] [-columns column_names]  
[-add_columns column_names]
```

Description

Use this command to report the scan elements that are created by the tool for scan insertion purposes.

The report is printed in a table format where each column represents an attribute specified by the attribute_name list. Only segment and leaf_cell type of scan_segments are reported unless specified by the object_spec argument. Use the -columns switch to specify which columns to show and the order in which they appear. The “name” column is mandatory and will always be shown. You can add additional columns after the default columns (or after the columns specified with the –columns switch) using the -add_columns switch.

Column names can be any of the following:

- Special Columns
 - si_so_clocks — the si,so clock domains. Each clock domain is preceded by +/- to indicate pos_edge/neg_edge accordingly.
- Scan Element Attribute Columns

All scan element built-in attributes listed in the “[Scan Element Object Type](#)” on page 3144“[Built in attributes on Scan Element \(SE\) Object Type](#)” on page 2.
- Module Attribute Columns

All module built-in attributes listed in “[Module](#)” on page 3058
- Instance Attribute Columns

All instance built-in attributes listed in “[Instance](#)” on page 3069

Arguments

- *object_spec*

An optional value that specifies a Tcl list of one or more object names or a collection of one or more scan element objects. Only existing scan_segment objects will be shown.

- **-columns *column_names***

An optional list of attributes or connector values to display for each scan element specified, including composite attributes such as “si_so_clocks”.

- **-add_columns *column_names***

An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

Example 1

The following example shows the default columns printed when the command is issued with no arguments.

```
report_scan_elements
```

name	type	length	si_so_clocks	state	is_non_scannable_reason
sch1/f5/SO1	segment	2	-clk1,-clk2	usable	--
sch2/f5/SO2	segment	2	+clk1,+clk3	usable	--
sch3/f6/so	segment	2	-clk2,+clk5	usable	--
/f	leaf_cell	1	+test_clk	ignored	inferred_from_drc
/f2	leaf_cell	1	+test_clk	ignored	user_specified
/f3	leaf_cell	2	-clk1	usable	--

Example 2

The following example includes user-specified columns.

```
report_scan_elements [get_scan_elements -state unusable] -columns {type id state is_valid}
```

name	type	id	state	is_valid
shift_register#/f4	shift_register	10	unusable	true
chain0	chain	12	unusable	true
inferred#bucket_0_1	inferred	13	unusable	true

Example 3

The following example adds additional columns to the default output using the -add_columns switch.

```
report_scan_elements -add_columns {type id used}
```

```
=====
name      type     length  si_so_clocks state   is_non_scannable_reason  id    used
-----  -----  -----  -----  -----  -----
sch1/f5/SO1 segment  2      -clk1,-clk2  usable   --                15   true
sch2/f5/SO2 segment  2      +clk1,+clk3  usable   --                16   true
sch3/f6/so  segment  2      -clk2,+clk5  usable   --                17   true
/f        leaf_cell 1      +test_clk   ignored  inferred_from_drc  19   false
/f2       leaf_cell 1      +test_clk   ignored  user_specified    20   false
/f3       leaf_cell 2      -clk1     usable   --                21   false
```

Example 4

The following example illustrates how you can define the columns of the report using a Tcl variable.

```
set cord { type state is_non_scannable_reason name}
report_scan_elements -column $cord [get_scan_element -state ignored]

=====
type      state   is_non_scannable_reason  name
-----  -----  -----  -----
leaf_cell  ignored  missing_scan_model    /tessent_persistent_cell_shift_capture_clock
leaf_cell  ignored  missing_scan_model    /tessent_persistent_cell_edt_clock
leaf_cell  ignored  imported_from_icl    /chip_top_gate_tessent_tap_main_inst/
leaf_cell  ignored  imported_from_icl    /chip_top_gate_tessent_tap_main_inst/instr_reg_2_
leaf_cell  ignored  imported_from_icl    /chip_top_gate_tessent_tap_main_inst/instr_reg_1_
```

Related Topics

[add_scan_segments](#)
[delete_scan_elements](#)
[get_scan_elements](#)
[report_scan_segments](#)

report_scan_enable

Context: dft -scan, dft -test_points

Mode: analysis, insertion

Reports on scan_enable signals and associated scan chains.

Usage

```
report_scan_enable [-scan_mode scan_mode] [<> | >>] file_pathname
```

Description

Reports on scan_enable signals and associated scan chains.

The following details are reported for each scan enable signal:

- Primary input port (top-level scan enable port)
- Internal connection node (internal instance pin designated as the scan enable signal driver)
- Associated scan chain(s)

Arguments

- **-scan_mode *scan_mode***

An optional switch and string pair that specifies the scan mode for which you want to tool to report information.

- **> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example reports the scan_enable details for all modes.

```
report_scan_enable
```

```
Scan mode 'int_mode' scan enables:  
-----  
Primary Input      Internal Connection Node  Scan Chain  
-----  
/spi_top/scan_en_out --                      int_mode_chain0  
                                         int_mode_chain1  
                                         int_mode_chain2  
                                         int_mode_chain3  
                                         int_mode_chain4  
  
Scan mode 'ext_mode' scan enables:  
-----  
Primary Input      Internal Connection Node  Scan Chain  
-----  
/spi_top/scan_en_out --                      ext_mode_chain0
```

Example 2

The following example reports the scan_enable details for int_mode only:

```
report_scan_enable -scan_mode int_mode  
Scan mode 'int_mode' scan enables:  
-----  
Primary Input      Internal Connection Node  Scan Chain  
-----  
/spi_top/scan_en_out --                      int_mode_chain0  
                                         int_mode_chain1  
                                         int_mode_chain2  
                                         int_mode_chain3  
                                         int_mode_chain4
```

Related Topics

[set_scan_enable](#)

report_scan_groups

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a report on all the current scan chain groups.

Usage

`report_scan_groups [{> | >>} file_pathname]`

Description

Displays a report on all the current scan chain groups.

The `report_scan_groups` command provides the following information in a report for each scan chain group:

- Name of the scan chain group
- Number of scan chains within the scan chain group
- Number of shifts
- Name of the test procedure file, which contains the information for controlling the scan chains in the reported scan chain group

Arguments

- `>file_pathname`

An optional redirection operator and pathname pair for creating or replacing the contents of `file_pathname`.

- `>>file_pathname`

An optional redirection operator and pathname pair for appending to the contents of `file_pathname`.

Examples

The following example displays a report of all the scan groups:

```
add_scan_groups group1 scanfile
add_scan_groups group2 scanfile1
report_scan_groups
```

Related Topics

[add_scan_groups](#)

[delete_scan_groups](#)

report_scan_models

Context: dft -scan

Mode: setup, analysis

Displays the sequential scan models currently in the scan model list.

Usage

```
report_scan_models
```

Description

Displays the sequential scan models currently in the scan model list.

The report_scan_models command displays sequential models which you previously added to the scan model list by using the [add_scan_instances](#) command.

Arguments

None

Examples

The following example displays all the sequential scan models from the scan model list:

```
add_scan_instances -modules d_flip_flop1 d_flip_flop2  
report_scan_models
```

report_scan_modes

Context: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Mode: setup

Reports all available scan modes after they are inserted by Tesson Scan.

Usage

```
report_scan_modes [-core core_name]
```

Description

Reports all available scan modes after they are inserted by Tesson Scan.

Arguments

- `-core core_name`

An optional switch that specify the core name for which the scan mode should be reported.

Examples

Example 1

The following example displays the scan modes before a scan mode was imported.

report_scan_modes

```
// command: report_scan_modes
//           Mode Name  Mode Type  EDT Instruments  OCC Instruments
//           -----  -----  -----  -----
//           int_edt_mode    internal      1          10
//           ext_mode        external      2
//           int_single_mode internal     10
```

Example 2

The following example displays the scan modes after a scan mode was imported.

report_scan_modes

```
//           Mode Name  Mode Type  EDT Instruments  OCC Instruments
//           -----  -----  -----  -----
//           int_edt_mode    internal      1          10
//           * ext_mode        external      2
//           int_single_mode internal     10
//           * = last imported scan mode
```

Related Topics

[import_scan_mode](#)

report_scan_polygons

Context: patterns -scan_diagnosis

Mode: setup, analysis

Traces scan chains and reports the cell and output net polygons for inter-scan cell logic located along the traces.

Usage

```
report_scan_polygons [-chain chain_name ...] [-start cell_number] [-stop cell_number]  
[-layout_marker marker_file] [-all_core_instances]
```

Description

The report_scan_polygons command traces from scan chain output to scan chain input for the specified chains and cell range. It reports the inter-scan cell logic found along the traces and the logics' bounding boxes. The report includes the logics' layers, critical areas, and polygons.

Scan cell numbering starts at 0 for the scan cell closest to the scan output and advances through the cells to the scan input. You can filter the report to include only specified chains and specified cells.

Refer to “[Inter-Scan Cell Polygon Reporting for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for more information.

Arguments

- **-chain *chain_name* ...**
An optional switch with strings that specifies a list of scan chain names as reported by the chain diagnosis report. By default, the tool reports all chains in the design.
- **-start *cell_number***
An optional switch and string pair that specifies to begin tracing inter-scan cell logic at the specified scan cell number. The default, 0, represents the scan cell closest to the scan output pin.
- **-stop *cell_number***
An optional switch and string pair that specifies to stop tracing inter-scan cell logic at the specified scan cell number. The default, -1, represents the scan cell closest to the scan input pin.
- **-layout_marker *marker_file***
An option switch and string pair that specifies to write a report in layout marker format supported by Calibre DESIGNrev.
- **-all_core_instances**
An optional switch that specifies to generate an inter-scan cell logic report with scanned net pathnames at the chip level. By default, the tool reports the scanned net pathnames at the

core level. This switch pertains to hierarchical designs and has no effect on non-hierarchical designs. Specifically, the tool generates the scanned net pathnames for each instance of a given core reported at the chip level. Refer to “[Inter-Scan Cell Polygon Reporting for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Manual*.

Examples

The following example reports the inter-scan cell logic for suspect 16 in the chain diagnosis report, which is suspect scan cell 333 on scan chain 78:

```
16      65  STUCK(IN+CELL+OUT) 1 /top_dut/io/data_out/SO FFTX1 \
top_dut/io/n18 333 chain78 /
```

Specifically, the following command specifies to trace and report the inter-scan cell logic between scan cell 333 and scan cell 334. The report shows that there is a buffer along the trace from scan cell 333 to scan cell 334.

```
report_scan_polygons -chain chain78 -start 333 -stop 334
```

For the scan cells and inter-scan cell logic, the report’s header lines list the identifier name, its gate type, library model name, and pin instance for each scan cell and connecting logic gate. For each listed cell and cell logic, it then reports the layer name (for nets) or “cell” (where “cell” is the bounding box of the cell listed in the header line), the critical area (for nets only), and the layout defect bounding box coordinates for the cell and output net.

```
333    LA  FD4TQHVTX1 /top_dut/io/data_out_regx0x/dff1
# The following lines list the polygons of the net connected to the
# scan output pin of cell 333
cell{ 67.76 291.2 73.64 295.12 }
M1 ca: 0.0378666 { 48.745 336.91 48.975 337.05 }
M1 ca: 0.0378666 { 67.83 292.905 67.97 293.135 }
M1 ca: 0.0740101 { 67.83 292.34 67.99 293.27 }
M1 ca: 0.0289322 { 49.17 337.05 49.4 337.22 }
M1 ca: 0.0703802 { 48.745 336.91 49.4 337.05 }
M3 ca: 1.24997 { 48.79 335.51 64.05 335.65 }
M3 ca: 0.438766 { 63.91 333.55 69.09 333.69 }
M3 ca: 0.1233 { 67.83 314.79 69.09 314.93 }
M3 ca: 0.0629425 { 63.865 333.55 64.375 333.69 }
M3 ca: 0.026492 { 63.865 335.51 64.095 335.93 }
M3 ca: 0.026492 { 48.745 335.51 48.975 335.93 }
M3 ca: 0.026492 { 68.905 333.27 69.135 333.69 }
M3 ca: 0.026492 { 68.905 314.51 69.135 314.93 }
M3 ca: 0.026492 { 67.785 314.51 68.015 314.93 }
BUF  BFHVTX1 /top_dut/io/iomuxc/iomuxc_registers/FE_OFC20932_n19
# The following lines list the polygons of the net connected to the
# scan output pin of the buffer, which also connects to the scan
# input pin of cell 333
cell{ 82.32 173.6 83.44 177.52 }
M1 ca: 0.1246 { 81.275 176.195 82.46 176.325 }
M1 ca: 0.0592873 { 71.705 292.39 72.215 292.53 }
M1 ca: 0.024477 { 81.225 176.19 81.455 176.61 }
M2 ca: 0.0556996 { 77.91 231.35 78.33 231.49 }
M2 ca: 0.5965 { 77.91 231.35 78.05 238.49 }
M2 ca: 0.0404092 { 77.07 240.265 77.21 240.495 }
M2 ca: 0.0404092 { 78.75 220.105 78.89 220.335 }
M2 ca: 0.0404092 { 77.63 220.105 77.77 220.335 }
M2 ca: 0.0629425 { 80.43 185.945 80.57 186.455 }
VIA2 ca: 0.0117878 { 77.635 272.235 77.765 272.365 }
VIA2 ca: 0.0117878 { 77.915 240.595 78.045 240.725 }
VIA2 ca: 0.0117878 { 77.075 240.315 77.205 240.445 }
VIA2 ca: 0.0117878 { 81.275 185.995 81.405 186.125 }
M3 ca: 0.100766 { 80.43 186.27 81.41 186.41 }
M3 ca: 0.0404092 { 77.585 272.23 77.815 272.37 }
M3 ca: 0.0404092 { 77.865 240.59 78.095 240.73 }
M3 ca: 0.026492 { 78.145 227.15 78.375 227.57 }
M3 ca: 0.026492 { 77.585 227.15 77.815 227.57 }
M3 ca: 0.026492 { 81.225 185.99 81.455 186.41 }
M3 ca: 0.026492 { 80.385 185.99 80.615 186.41 }
334    LA  FD4TQHVTX1 /top_dut/io/data_out_regx0x/dff1
# The following lines list the polygons of the net connected to the scan
# output pin of cell 334, which also connects to the input of the
# buffer
cell{ 84.84 173.6 90.72 177.52 }
M1 ca: 0.1675 { 83.3 175.355 84.98 175.485 }
M1 ca: 0.0740101 { 84.91 174.74 85.07 175.67 }
M1 ca: 0.0202047 { 83.095 175.025 83.37 175.535 }
...
```

Related Topics

[get_layout_core_instance](#)

set_layout_core_instance

report_scan_segments

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all modes

Reports the scan segment declarations that are defined either via add_scan_segments command or tcd_scan files.

Usage

```
report_scan_elements [object_spec] [-columns column_names]  
[-add_columns column_names]
```

Description

Use this command to report the scan segment declarations that are defined either via add_scan_segments command or tcd_scan files.

The report is printed in a table format where columns represent the scan segment specifications, such as length, traceability, pin names on the segment container and so forth. The inversion information of the pins is represented with a ‘~’ in front of the pin names in the table.

Note

 Pin columns will show the full pin path in the event that the pin path is different than the name path or “.” to indicate the path is the same as the path of the scan element as indicated in the name field.

Column names can be any of the following:

- Special Columns
 - from_tcd — the from_tcd scan segment property.
 - reset_pins — the active_high_reset pin(s) (preceded by “~” if polarity is inverted).
 - scan_en_pins — the scan_enable pin(s) (preceded by “~” if polarity is inverted).
 - set_pins — the active_high_set pin(s) (preceded by “~” if polarity is inverted).
 - si_pin — the scan_in pin(s) (preceded by “~” if polarity is inverted).
 - si_so_clock_pins — the clock_in,clock_out pins (each clock is preceded by “~” if polarity is inverted).
 - si_so_clocks — the si,so clock domains. Each clock domain is preceded by +/- to indicate pos_edge/neg_edge accordingly.
 - ste_pins — the test_enable pin(s) (preceded by “~” if polarity is inverted).
 - so_pin — the scan_out pin(s) (preceded by “~” if polarity is inverted)
- Scan Element Attribute Columns

All scan element built-in attributes listed in the “[Scan Element Object Type](#)” on page 3144“Built in attributes on Scan Element (SE) Object Type” on page 2.

- Module Attribute Columns
All module built-in attributes listed in “[Module](#)” on page 3058
- Instance Attribute Columns
All instance built-in attributes listed in “[Instance](#)” on page 3069

Arguments

- *object_spec*
An optional value that specifies a Tcl list of one or more object names or a collection of one or more scan element objects. Only existing scan_segment objects will be shown.
- **-columns** *column_names*
An optional list of attributes or connector values to display for each scan element, including composite attributes such as “si_so_clocks”.
- **-add_columns** *column_names*
An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

The following example shows the default columns printed when the command is issued with no arguments.

report_scan_segments

Name	length	traceable	si_pin	so_pin	si_so_clocks	si_so_clock_pins	scan_en_pins	set_pins	reset_pins	ste_pins
s1_ch1/s1/so1 5	yes		./si1	./so1	—	./clk1,./clk1	./se	—	—	—
s2_ch2/s1/so2 5	no		./si2	./so2	—	./clk3n,./clk3n	./clk3n,./clk3n	./se	—	—

Related Topics

- [add_scan_segments](#)
- [delete_scan_elements](#)
- [get_scan_elements](#)
- [report_scan_elements](#)

report_scan_volume

Context: dft -edt, patterns -scan

Mode: analysis

Displays the volume of scan data used by the pattern set.

Usage

```
report_scan_volume [-External] [<> | >>] file_pathname]
```

Description

Displays the volume of scan data used by the pattern set.

The report_scan_volume command displays the volume of scan data used by the pattern set, including chain test patterns. The report format depends on the design and type of patterns, as illustrated in the examples. Because the pattern count is not a good measure of EDT compression for different scan configurations, this command gives you a metric that is equally good for EDT and regular ATPG, making it easier to compare their results.

Arguments

- `-External`

An optional switch that displays the volume of scan data used by the external pattern set. Prior to using this switch, you must specify an external pattern set using the [read_patterns](#) command.

- `>file_pathname`

An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- `>>file_pathname`

An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

This basic EDT design example and the examples that follow show the volume display for various types of designs and patterns.

$$\text{Scan Volume} = (\# \text{ Scan Channels}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Scan Loads})$$

```
// -----
// Scan volume report.
// -----
//   channels : 1
//   shift cycles : 371+32
// -----
//   pattern   # test   # scan           volume
//   type     patterns   loads (cell loads or unloads)
// -----
// chain test      6       6             2418
```

Example 2

The following example is a basic EDT design where Ram_sequential or multiple-load patterns trigger an expanded format because of a difference in input and output volume:

$$\text{Input Volume} = (\# \text{ Scan Channels}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Scan Loads})$$

$$\text{Output Volume} = (\# \text{ Scan Channels}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Test Patterns})$$

```
// -----
// Scan volume report.
// -----
//   channels : 1
//   shift cycles : 371+32
// -----
//   pattern   # test   # scan   input volume   output volume   average volume
//   type     patterns   loads   (cell loads)   (cell unloads)
// -----
// chain test      6       6         2418          2418          2418
// basic          34      34        13702         13702         13702
// clock_po       2        2         806            806            806
// ram_seq        57      223       89869         22971         56420
// clock_seq      304     304       122512        122512        122512
// multi_load    12       24        9672          4836          7254
// -----
// total          415     593       238979        167245        203112
```

Example 3

The following example is an EDT design where the number of input channels differs from the number of output channels:

$$\text{Input Volume} = (\# \text{ Input Channels}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Scan Loads})$$

$$\text{Output Volume} = (\# \text{ Output Channels}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Test Patterns})$$

```
// -----
// Scan volume report.
// -----
//      input channels : 8
//      output channels : 9
//      shift cycles   : 30
// -----
//      pattern    # test  # scan  input volume  output volume  average volume
//      type       patterns loads   (cell loads) (cell unloads)
// -----
//      chain test     4      4        960          1080         1020
//      basic        80     80      19200         21600        20400
//      ram_seq       31     118      28320         8370         18345
//      clock_seq     85     85      20400         22950        21675
//      multi_load    10     30       7200          2700         4950
// -----
//      total         210    317      76080         56700        66390
```

Example 4

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design; no Ram_sequential or multiple-load patterns:

$$\text{Scan Volume} = (\# \text{ Scan Chains}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Scan Loads})$$

```
// -----
// Scan volume report.
// -----
//      scan chains : 10
//      shift cycles : 31
// -----
//      pattern    # test  # scan           volume
//      type       patterns loads   (cell loads or unloads)
// -----
//      chain_test     1      1            310
```

Example 5

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design with Ram_sequential or multiple-load patterns:

$$\text{Input Volume} = (\# \text{ Scan Chains}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Scan Loads})$$

$$\text{Output Volume} = (\# \text{ Scan Chains}) \times (\# \text{ Shift Cycles}) \times (\# \text{ Test Patterns})$$

```
// -----
// Scan volume report.
// -----
//      scan chains : 10
//      shift cycles : 31
// -----
//      pattern    # test   # scan   input volume   output volume   average volume
//      type      patterns   loads   (cell loads)   (cell unloads)
// -----
//      chain test      1       1           310           310           310
//      basic          42      42        13020        13020        13020
//      clock_po       1       1           310           310           310
//      ram_seq        6       18          5580          1860          3720
//      clock_seq      53      53        16430        16430        16430
//      multi_load     3       6           1860          930          1395
// -----
//      total         106     121        37510        32860        35185
```

Related Topics

[report_edt_configurations](#)
[report_edt_pins](#)
[set_edt_options](#)

report_seq_transparent_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Displays a list of seq_transparent test procedures along with the associated data that you specify.

Usage

```
report_seq_transparent_procedures [-All | procedure_name...] [-CElls] [-Load_disturbs]
[-CAapture_disturbs] [{> | >>} file_pathname]
```

Description

Displays a list of seq_transparent test procedures along with the associated data that you specify.

The report_seq_transparent_procedures command displays a list of the specified seq_transparent test procedures. You can optionally display data associated with each seq_transparent test procedure by specifying the appropriate switches.

If you do not specify any of the data switches, the command does not provide any details about the procedures.

Arguments

- **-All**
An optional switch that displays the associated data for all seq_transparent procedures. If you do not specify any other argument with the -All switch, the command simply lists the names of all the currently defined seq_transparent procedures. This is the default.
- **procedure_name**
An optional repeatable string that specifies the names of the seq_transparent procedures whose data you want to display.
- **-CElls**
An optional switch that displays the seq_transparent cells associated with the specified procedures. By default, the tools do not display these cells.
- **-Load_disturbs**
An optional switch that displays the scan cells with load disturbs associated with the specified procedures. By default, the tools do not display these cells.
- **-CAapture_disturbs**
An optional switch that displays the scan cells, primary outputs, seq_transparent cells, and RAMs which had capture disturbs. By default, the tools do not display these items.

- **>file.pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file.pathname*.

- **>>file.pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file.pathname*.

Examples

The following example displays all the seq_transparent test procedures, along with the associated scan cells that had load disturbs:

```
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 clk
set_system_mode analysis
report_seq_transparent_procedures -all -load_disturbs
```

Related Topics

[report_gates](#)

report_sequential_fault_depth

Context: dft -edt, patterns -scan

Mode: analysis

Displays estimates of maximum test coverage possible with different sequential depth settings.

Usage

```
report_sequential_fault_depth [<> | >>] file_pathname]
```

Description

Displays estimates of maximum test coverage possible with different sequential depth settings.

The report_sequential_fault_depth command displays the maximum test coverage the tool estimates is possible if all relevant collapsed faults are detectable. Use this command to quickly assess the upper limits of coverage possible under optimal test conditions for various sequential depths.

The data is displayed in three columns as follows:

- Depth numbers for fifteen successive sequential depths, starting with the current depth setting. The default sequential depth upon invocation of the tool is zero, but you can change it using the -Sequential switch with the [set_pattern_type](#) command.
- Number of targeted faults for each depth
- Estimated maximum collapsed test coverage achievable under the assumption that all relevant collapsed faults can be detected. This figure is an upper bound of the coverage you can expect.

Note



Relevant faults include those the tool declared ATPG_untestable (AU) in a previous ATPG run—because the tool will retarget these faults when you increase the depth. Accordingly, the tool includes these AU faults in the row corresponding to the current depth+1.

Arguments

- >*file_pathname*

An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >>*file_pathname*

An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example shows the output of the report_sequential_fault_depth command for a sequential depth of 4:

report_sequential_fault_depth

```
// Structural heuristic analysis of the sequential depth of faults
//
//   Set      Number      Max.
// Depth    of target    possible
//          to        faults    TC[%]
//          this      (coll.)  (coll.)
//          4         1450     89.37
//          5         7002     93.59
//          6         7852     94.42
//
//          ...
//          15        11897    98.34
//          16        12033    98.47
//          17        12099    98.53
//          18        12159    98.59
```

Related Topics

[set_pattern_type](#)

report_shift_registers

Context: dft -scan, dft - test_points

Mode: setup, analysis

Reports the identified shift registers in the design after switching to analysis mode.

Usage

```
report_shift_registers [-Verbose | -Summary]
```

Description

Reports the identified shift registers in the design after switching to analysis mode.

The tool tries to preserve the original connections inside the identified shift registers during stitching. Therefore, this command may report the shift registers in the design differently, before and after the execution of the insert_test_logic command.

For each identified shift register, this command reports the following information:

- ID, assigned by the tool
- Length
- First and last flip-flop properties (instance pathname, clock edge and name, library model name) unless the -verbose switch is specified in which case the properties of all flip-flops in the shift registers are reported

Arguments

- **-Verbose**
An optional switch that reports all of the flip-flops identified in the shift registers.
- **-Summary**
An optional switch that reports a summary text without printing the flip-flops identified in the shift registers.

Examples

Example 1

The following example shows the output when neither switch is specified:

report_shift_registers

Id	Length	SequentialCell	Clock	Library
		Instance PathName	Edge & Name	ModelName
[1]	4	/ud1 /ud4	+ clk + clk	dff dff
[2]	2	/uA/uf9 /uA/uf10	- clk - clk	dff dff
// Number of sequential elements in design: 10 // Number of shift register flops recorded for scan insertion: 6 // => 60% of all sequential elements in design // Number of shift registers recorded for scan insertion: 2 // Longest shift register has 4 flops. // Shortest shift register has 2 flops. // Potential number of nonscan flops to be converted to scan cells: 2 // Potential number of scan cells to be converted to nonscan flops: 0				

Example 2

The following example shows the output when the -verbose switch is specified:

report_shift_registers -verbose

Id	Length	SequentialCell	Clock	Library
		InstancePathName	Edge & Name	ModelName
[1]	4	/ud1 /ud2 /ud3 /ud4	+ clk + clk + clk + clk	dff dff dff dff
[2]	2	/uA/uf9 /uA/uf10	- clk - clk	dff dff
// Number of sequential elements in design: 10 // Number of shift register flops recorded for scan insertion: 6 // => 60% of all sequential elements in design // Number of shift registers recorded for scan insertion: 2 // Longest shift register has 4 flops. // Shortest shift register has 2 flops. // Potential number of nonscan flops to be converted to scan cells: 2 // Potential number of scan cells to be converted to nonscan flops: 0				

Example 3

The following example shows the output when the -summary switch is specified:

report_shift_registers -summary

```
// Number of sequential elements in design: 6
// Number of shift register flops recorded for scan insertion: 4
// => 66.67% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 1
// Longest shift register has 4 flops.
// Shortest shift register has 4 flops.
// Potential number of nonscan flops to be converted to scan cells: 1
// Potential number of scan cells to be converted to nonscan flops: 0
```

Related Topics

[set_shift_register_identification](#)

report_silicon_insight_result

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Reports the summary diagnosis results for failing flops and pseudo failing flops for ATPG.

Usage

```
report_silicon_insight_result -atpg_datalog {scan_cells | patterns | full}
```

Arguments

- **-atpg_datalog {scan_cells | patterns | full}**

A required switch and string choice that specifies the type of summary information to return:

scan_cells—Reports a list of failing cells and pattern counts for the failing cells. This type of report is useful for determining how often each cell fails in all failing patterns.

patterns—Reports a list of all failing patterns and their failing flop/cell count per failing pattern.

full—Reports a list of all failing patterns, their failure counts, and a description of each failing cell in the pattern.

Examples

Example 1

The following example displays the format for the scan_cells summary diagnosis results. For failing flops, the report displays the cell name, channel ID, and cell index. For failing_pseudo_flops, the report displays the channel ID and cell index.

```
// Cells summary: Summary report for failing patterns count per cell:  
// -----  
// | CellName | ChannelId | CellIndex | PatternCount |  
// -----  
// | PIN | SFRWE_c2_2 | 0 | 80 |  
// | my_glue/cpu_i/uPORT/ix1734 | edt_channel13 | 15 | 1 |  
// | my_glue/cpu_i/uPORT/ix687 | edt_channel13 | 17 | 1 |  
// | my_glue/cpu_i/uPORT/ix1754 | edt_channel13 | 18 | 1 |  
// | my_glue/cpu_i/uPORT/ix1794 | edt_channel13 | 1 | 1 |  
// | my_glue/cpu_i/uPORT/ix1814 | edt_channel13 | 2 | 1 |  
// | my_glue/cpu_i/uPORT/ix1834 | edt_channel13 | 3 | 1 |  
// | my_glue/cpu_i/uPORT/ix1854 | edt_channel13 | 4 | 1 |  
// | my_glue/cpu_i/uPORT/ix701 | edt_channel13 | 10 | 1 |  
// | my_glue/cpu_i/uPORT/ix729 | edt_channel13 | 12 | 1 |
```

Example 2

The following example displays the format for the patterns summary diagnosis results.

```
// Patterns summary: Summary report for failing flop frequency per pattern:  
// -----  
// | Pattern# | FlopCount |  
// -----  
// | 0 | 1 |  
// | 1 | 4 |  
// | 2 | 5 |  
// | 3 | 6 |  
// | 4 | 1 |
```

Example 3

The following example displays the format for the full summary diagnosis results. For failing flops, the report displays the cell name, channel ID, and cell index. For failing_pseudo_flops, the report displays the channel ID and cell index.

```
// Full report: Full report for failing patterns:  
// -----  
// | Pattern# | CellName | ChannelId | CellIndex | Chain | CellId |  
// -----  
// | 0 | PIN | SFRWE_c2_2 | 0 | | |  
// | 1 | PIN | SFRWE_c2_2 | 0 | | |  
// | 1 | my_glue/cpu_i/uPORT/ix1734 | edt_channel3 | 15 | chain18 | 15 |  
// | 1 | my_glue/cpu_i/uPORT/ix687 | edt_channel3 | 17 | chain18 | 17 |  
// | 1 | my_glue/cpu_i/uPORT/ix1754 | edt_channel3 | 18 | chain18 | 18 |  
// | 2 | PIN | SFRWE_c2_2 | 0 | | |  
// | 2 | my_glue/cpu_i/uPORT/ix1794 | edt_channel3 | 1 | chain19 | 1 |  
// | 2 | my_glue/cpu_i/uPORT/ix1814 | edt_channel3 | 2 | chain19 | 2 |  
// | 2 | my_glue/cpu_i/uPORT/ix1834 | edt_channel3 | 3 | chain19 | 3 |  
// | 2 | my_glue/cpu_i/uPORT/ix1854 | edt_channel3 | 4 | chain19 | 4 |
```

report_simdut_faults

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Reports the SimDUT faults.

Usage

report_simdut_faults

Arguments

None

Examples

The following example reports a list of active stuck-at faults stored in the simulator:

```
report_simdut_faults

// Net name | fault type
// -----
// /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][1] /Q | 1
// /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][2] /Q | 0
// /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][4] /Q | 1
```

Related Topics

[add_simdut_fault](#)

[delete_simdut_fault](#)

report_simulation_contexts

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Lists the available simulation contexts and indicates the current simulation context.

Usage

```
report_simulation_contexts [-predefined | -user_defined]
```

Description

Lists the available simulation contexts and indicates the current simulation context.

The report_simulation_contexts command lists all simulation contexts. Note that the command reports available simulation contexts, some of which are only available after the tool reads a procedure file.

If you issue the command without arguments, the command reports all simulation contexts as follows:

```
Simulation Context Report

Name           Class      Populated
-----
empty_context   User-defined yes
stable_after_setup Predefined no
stable_capture   Predefined no
* stable_load_unload Predefined no
stable_shift     Predefined no
tie_value        Predefined yes

* = current simulation context
```

The Populated column reports with a “yes” only those contexts that have correct simulation values. The simulation values are created in the DRCs.

Arguments

- **-predefined**

An optional switch that reports predefined simulation contexts, which are listed in [Table 6-7](#) on page 2023.

- **-user_defined**

An optional switch that reports user-defined simulation contexts.

Related Topics

[add_simulation_context](#)

[get_simulation_context_list](#)
[add_simulation_forces](#)
[get_simulation_value_list](#)
[copy_simulation_context](#)
[report_simulation_forces](#)
[delete_simulation_contexts](#)
[set_current_simulation_context](#)
[delete_simulation_forces](#)
[simulate_clock_pulses](#)
[get_current_simulation_context](#)
[simulate_forces](#)

report_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Lists the active forces on the specified gate_pin objects.

Usage

report_simulation_forces [*obj_spec*]

Description

Lists the active forces on the specified gate_pin objects.

The report_simulation_forces command lists the simulation forces on the specified gate_pin object(s). When you issue this command without arguments, the tool lists all simulation forces of the current simulation context.

Arguments

- *obj_spec*

An optional value that specifies one or more gate_pin objects.

Examples

The following example shows the output of the report_simulation_forces command:

```
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 0
ANALYSIS> add_simulation_forces [get_gate_pins I2] -value Z
ANALYSIS> add_simulation_forces [get_gate_pins reg12/Q] -value 1
ANALYSIS> report_simulation_forces
```

Gate Pin	Gate Pin ID	Value
'Din1[0]'	7.0	0
'Din1[1]'	6.0	Z
'Din1[2]'	5.0	1

Related Topics

[add_simulation_context](#)
[get_simulation_context_list](#)
[add_simulation_forces](#)
[get_simulation_value_list](#)
[copy_simulation_context](#)
[report_simulation_contexts](#)
[delete_simulation_contexts](#)

[simulate_clock_pulses](#)
[delete_simulation_forces](#)
[simulate_forces](#)

report_simulation_library_sources

Context: unspecified, all contexts

Mode: all modes

Reports the values specified with the set_simulation_library_sources command.

Usage

```
report_simulation_library_sources
```

Description

Reports the values specified with the [set_simulation_library_sources](#) command.

Arguments

None.

Examples

The following example uses the [set_simulation_library_sources](#) to define search paths, and to map a pre-compiled library name to its physical directory. The report_simulation_library_sources command is then used to report them.

```
set_simulation_library_sources \
  -y ..//verilog ..//data/customPads \
  -v ..//data/extralib.v \
  -extensions .v \
  -logical_library_map_list {mylib ..//data}
report_simulation_library_sources

// Simulation Library Source Paths
// =====
// type    path                      file extensions
// -----  -----
// dir     '../verilog'              'v v.gz'
// dir     '../..//data/customPads'   '.v.gz'
// file   '../data/extralib.v'
//
// Simulation Library Logical Library Map
// =====
// name    dir path
// -----  -----
// mylib   ..//data
```

Related Topics

[set_simulation_library_sources](#)

report_static_dft_signal_settings

Context: dft patterns

Mode: setup, analysis, insertion

Prerequisites: ICL is elaborated

Reports the setting specified or implied with the [set_static_dft_signal_values](#) command.

Usage

```
report_static_dft_signal_settings  
[-all | -current_design | -icl_instances icl_instances | -instances instances]
```

Description

Reports the settings specified or implied with the [set_static_dft_signal_values](#) command. All static DFT signals implemented with IJTAG can be set using the [set_static_dft_signal_values](#) command. See the [add_dft_signals](#) command for information on how to add the DFT signals. The DFT signals are grouped by each module in which you added the signals and ran the [process_dft_specification](#) command. These modules have the `tessent_design_level` attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value following the logic described in the [set_static_dft_signal_values](#) command description section.

Arguments

- **-all**

An optional switch that reports the settings that exist on DFT signals found within all ICL instances that have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also reported.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. If unspecified, the `-current_design` switch is the default.

- **-current_design**

An optional switch that reports the settings that exist on DFT signals found within the top module. The top-level module must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within the ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. If unspecified, the `-current_design` switch is the default.

- **-icl_instances *icl_instances***

An optional switch that reports the settings that exist on DFT signals found within the specified ICL instances. These ICL instances must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

- -instances *instances*

An optional switch that reports the settings that exist on DFT signals found within the specified design instances. These design instances must match the value of the “hierarchical_design_instance” attribute found on an ICL instance with the tesson_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

Examples

The following example shows the report generated for a block where the [set_static_dft_signal_values](#) was explicitly called for the ltest_en DFT signal. The report shows the inferred values on the other signals.

```
// ICL Module      : sub_block_13

// -----  -----
// DFT Signal Name Usage          Set value Set source
// -----  -----
// all_test        global_dft_control 1       Inferred
// ext_ltest_en    logic_test_control -       -
// int_ltest_en    logic_test_control -       -
// ltest_en        logic_test_control 1       Explicit
// ext_multi       scan_mode(external) -       -
// int_edt         scan_mode(internal) -       -
```

Related Topics

[add_dft_signals](#)
[set_static_dft_signal_values](#)

report_statistics

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis, insertion (dft -scan context only)

Displays a detailed statistics report to the screen.

Usage

```
report_statistics
  [[-DEtailed_analysis [ DI ] [-Threshold threshold_number]]
   [-Instance instance_pathname | -Model {ALL | model_name} |
    -CLOCK_domains {ALL | SUMMARY | clock_pathname...} | |
    -Hierarchy [ -FORmat “format_code ... ”]
      [-Sort “format_code” [-Ascending | -Descending]]
      [-EXclude_unlisted_faults | -Grayboxes_only]
      [-LESsthan percentage]
      [-LEVel integer]
      [-DETections integer]
      [-INDent integer | [-USe_full_instance_path]
      [-CSV]]]
   [-DISplay]
   [(> | >>) file_pathname]
```

Description

Displays a detailed statistics report to the screen.

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

When you use the -Instance switch, the tool's statistics report lists the following three groups of information:

- The current number of collapsed or total faults, depending on the current fault mode, in each class. By default, the fault mode is full. The report does not display fault classes with no members.
- The percentage of test coverage, fault coverage, fault coverage loss, and ATPG effectiveness for collapsed or total faults. By default, the fault mode is full, however the tool can report either the collapsed number or full number depending on the current fault mode.
- The total numbers for the following:
 - Total patterns simulated in the preceding fault simulation process. This subgroup may additionally contain total numbers for the following internal patterns sets:
 - Basic scan patterns
 - Clock_po patterns

- Ram_sequential patterns
- Clock_sequential patterns
- Total patterns currently in the test pattern set
- Total CPU time

If a pattern type has no patterns, the report does not display the count for that type. If all patterns are basic patterns, it will not display any count. And it counts clock_sequential patterns that are also clock_po only as clock_sequential patterns.

When you use the -Hierarchy switch, the statistics report lists the following two groups of information for each reported hierarchical instance:

- Instance leaf name (add -USe_full_instance_path switch to report entire instance path)
- A customizable list of statistics that, by default, consists of the following:
 - Module name
 - Total number of faults
 - Number of unobserved (UO) faults
 - Number of ATPG untestable (AU) faults
 - Test coverage

When you issue the “set_atpg_timing On” command prior to create_patterns, the statistics report also includes delay test coverage. For more information about how the tool calculates delay test coverage, refer to the *Tessent Scan and ATPG User’s Manual*.

Arguments

- -DEtailed_analysis [DI] [-Threshold *threshold_number*]An optional switch that, by default, reports detailed information for the following fault sub-classes:
 - AU.CC (CELL_CONSTRAINTS)
 - AU.PC (PIN_CONSTRAINTS)
 - AU.TC (TIED_CELLS)
 - UD.AAB (ATPG_ABORT)
 - UD.EAB (EDT_ABORT)
 - UD.UNS (UNSUCCESS)

For information about any of these fault sub-classes, refer to the discussion about “[Fault Sub-classes](#)” in the *Tessent Scan and ATPG User’s Manual*.

For an example of reporting detailed statistics for AU or DI sub-classes, refer to “[Example 2](#)” on page 1793.

DI

An optional literal that enables the tool to report the DI fault sub-classes without performing the detailed analysis for AU faults, which can be time consuming. All of the DI sub-classes are listed under “[Fault Sub-classes](#)” in the *Tessent Scan and ATPG User’s Manual*. Note that the tool always performs analysis for all DI fault sub-classes, with the exception of DI.EDT, which you control using the [set_fault_subclass_analysis](#) command. For more information about DI.EDT analysis, refer to that command description.

-Threshold *threshold_number*

An optional switch and numerical pair that limits the list of untestable faults displayed. The *threshold_number* is the minimum coverage loss percentage to display. The default threshold is 0.1%. Faults having a coverage loss less than the threshold are summarized at the end of the list (see “[Example 2](#)” on page 1793).

- -Instance *instance_pathname*

An optional switch and string pair that reports fault statistics for a specific instance. The *instance_pathname* is the name of a circuit block whose statistics you want to report. Only fault statistics are affected by this option; pattern count statistics apply to the entire design.

- -Model ALL | *model_name*

An optional switch and string pair that specifies reporting statistics for all models or a single named model in a Tessent Cell library. When specifying a single model, *model_name* must match a cell model in the Tessent Cell library. This switch is only available for Tessent FastScan and Tessent TestKompress when invoked in model mode (-model). If you use this switch in other than model mode, the tool issues the following error:

Error: -model reporting only available for -model tool invocations.

- -CLOCK_domains SUMMARY | ALL | *clock_pathname...*

An optional switch and literal or repeatable string pair that you use for the following:

- Reporting the summary of clock domain test coverage; or
- Specifying a list of clocks to report the coverage(s) for the specified clock domain(s).

Choose from the following options:

SUMMARY — A literal that reports a summary of clock domain coverage and produces a Clock Domain Summary section. For each clock domain, the tool reports the percentage of faults in the clock domain with respect to the total fault population, and reports the relevant test coverage numbers for each clock domain.

ALL — A literal that reports the detailed coverage information for all clock domains.

clock_pathnames... — A repeatable string specifying a clock domain or clock domains. The tool reports the detailed coverage information for the specified clock domains.

- **-Hierarchy**

An optional switch that allows fault statistics to be reported for each hierarchical instance in the design. The tool uses columns for all fields reported. For each reported instance, the tool displays the instance name and flags the instance if it is a gray box (GB) or black box (BB). If an instance is a gray box, the tool adds rows reporting statistics of faults for “Instances in netlist” and “Instances not in netlist”. By default, the tool lists the following statistics for each reported instance:

- Module name
- Total number of faults
- Number of unobserved (UO) faults
- Number of ATPG untestable (AU) faults
- Test coverage
- Test coverage loss

-FORmat “format_code...”

An optional switch and a string of one or more format codes that, together with the **-Hierarchy** switch, specifies which data to display and the order in which to display it. Use this switch to tailor the content and arrangement of displayed data in hierarchical reports. By default, only the instance column is displayed when using the **-format** switch. [Table 5-17](#) lists the format code literals this switch recognizes and the information displayed for each.

Table 5-17. Format Codes

Tessent FastScan Information (uncollapsed)	Format Code	Report Label
Fault coverage	%fc	Fault Coverage
Test coverage	%tc	Test Coverage
Test coverage loss	%tcl	TC Loss
Relevant test coverage	%rtc	Relevant TC
ATPG effectiveness	%ae	ATPG Eff.
Module name	mo	Module
Number of faults	#f	# faults
Fault category	<category>[.<subclass>]	<CAT>[.<SUB>]

The **-format** option reports statistics for fault subclasses specified using format codes with this format:

<fault category>[.<subclass>]

For example:

```
report_statistics -hierarchy -format "%tc %rtc AU AU.BB" -level 1
Hierarchical Statistics Report
Stuck-at Faults
-----
Instance          Test Coverage Relevant TC   AU    AU.BB
-----
-top-            59.12%      59.12%    14465    116
CHIPJ_MGsg_INT_CD_FLOP_DECODER_INST 79.41%      79.41%      7      0
DEF              89.85%      89.85%    166      0
LVISION_JTAP_INST        0.00%      0.00%      0      0
LVISION_LOGICTEST_INST       0.00%      0.00%      0      0
LV_BURST_CLK_CTRL_I1        0.00%      0.00%      0      0
LV_BURST_CLK_CTRL_I2        0.00%      0.00%      0      0
LV_SE_CTRL_BL5_P_I1        77.34%      77.34%    29      0
LV_SE_CTRL_BL5_P_I2        6.83%       76.83%    19      0
LV_SHIFT_CLK_CTRL_INST      0.00%      0.00%      0      0
TLB_inst           57.69%      57.69%    4100    116
```

All format codes, including the fault codes, are case insensitive. For example:

```
report_statistics -hierarchy -format "%fc au Au.UNC Di.sCAn, DI.UNC"
```

When duplicated format codes exist in the format code list, only one appears in the reported statistics.

Information displays in the same order as the format codes in the command. The default for this switch is:

```
-format "mo #f uo au %tc"
```

The results display as shown in the following example:

```
Hierarchical Statistics Report
Stuck-at Faults
-----
Instance     Module     #faults   UO      AU    Test Coverage
-----
-top-       my_core    46250    2316    768    93.16%
u10        m3s018bo   6240     1038    99     81.36%
u11        m3s015bo   4526     122     60     95.97%
u12        m3s016bo_3  590      7       8      97.46%
u1         m3s020bo   2146     87     33     94.40%
```

Note

 You must include a space between each format code you specify, and begin and end the sequence of format codes with a double quotation mark ("").

-Sort *format code* [-Ascending | -Descending]

An optional switch and literal pair that allows the tool to sort the results by the column indicated by *format code*. Sorting begins with level 1 instances, proceeds

with each level 1 instance's child (level 2), and continues hierarchically until complete. If -sort is not used, the report is sorted by the instance names in ascending alphabetical order.

-Ascending

An optional switch to sort in ascending order.

-Descending

An optional switch to sort in descending order. This is the default sort order.

The -sort option parser supports fault <category>[.<subclass>] format codes, case insensitive format codes, and will ignore duplicated format column codes.

-EXClude_unlisted_faults

An optional switch that excludes unlisted faults from the report and calculations. It also omits the "Instances in netlist" and "Instances not in netlist" lines from the report.

-Grayboxes_only

An optional switch that together with the -Hierarchy switch, limits the report to graybox instances, not including the top instance. This option automatically enables the "-use_full_instance_path" option.

When you use this switch, the header of the statistics report includes the text "(Grayboxes Only)".

-LESsthan *percentage*

An optional switch and real number pair that, together with the -hierarchy switch, specifies to report only module instances with test coverage less than or equal to the specified percentage. The default percentage is 100.0 and causes all module instances to be reported.

Specifying "-lessthan 90.0", as shown in this example, the tool reports statistics only for module instance u10 because u10 is the only instance with test coverage less than or equal to 90%:

```
report_statistics -hierarchy -lessthan 90.0
```

```
Hierarchical Statistics Report
Stuck-at Faults
-----
Instance      Module      #faults    UO      AU  Test Coverage
-----        -----       -----      --   ---   -----
u10          m3s018bo    6240       1038     9    81.36%
```

-LEVel *integer*

An optional switch and integer pair that, together with the -Hierarchy switch, controls the hierarchy reporting depth. The tool reports all hierarchical levels down to and including the level you specify, counting from the top level, which is 0. If you specify "-level 1", for example, the tool will display data for the top level and the instances one level of hierarchy beneath the top level. By using this switch to restrict reporting

to the upper levels of a design's hierarchy, you can reduce the command's processing time on very large designs with deep hierarchy. The maximum allowed value is 1000.

-DETectors *integer*

An optional switch and integer pair used with the -Hierarchy switch to represent the number of multiple detections. When specified, the DS fault count, test coverage, relevant coverage, fault coverage, test coverage loss, and ATPG efficiency metrics adjust to reflect the values with detections greater than or equal to the detection number specified.

By default, the tool runs as though you had specified “-detections 1”.

-INDent *integer*

An optional switch and integer pair that, together with the -hierarchy switch, specifies the number of additional spaces to indent the information for each subsequent level of reported hierarchy. The default indentation is two spaces. Specifying “-indent 0” removes all indentation. The maximum allowed value is 10.

-USe_full_instance_path

An optional switch used to display the complete instance path. Without it, the report displays the leaf instance name and the instantiation level is shown by indentation.

-CSV

An optional switch that uses a comma as a delimiter between the columns in the report, making the data readily usable within spreadsheet programs. This option automatically enables the “-use_full_instance_path” option.

When you use this switch, the statistics report has no header.

- -Display

An optional switch that specifies to display the following columns in the DFTVisualizer Instance Browser: Test Coverage, Test Coverage Loss, Total Faults, AU, and Relevant Test Coverage.

- > *file_pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays a statistics report for the top level instance in a design after performing an ATPG run:

```
set_system_mode analysis  
create_patterns  
report_statistics
```

```
Statistics Report  
Transition Faults  
-----  
Fault Classes  
-----  
Fault Classes #faults #faults  
              (total) (total relevant)  
-----  
    FU (full)        1446    1424  
-----  
    DS (det_simulation)    24 ( 1.66%) same ( 1.69%)  
    DI (det_implication)   326 (22.54%) same (22.89%)  
    UU (unused)          242 (16.74%) same (16.99%)  
    AU (atpg_untestable)  854 (59.06%) 832 (58.43%)  
-----  
Fault Sub-classes  
-----  
    AU (atpg_untestable)  
    BB (black_boxes)      176 (12.17%) same (12.36%)  
    PC* (pin_constraints)  22 ( 1.52%) deleted  
    TC* (tied_cells)      120 ( 8.30%) same ( 8.43%)  
    Unclassified          536 (37.07%) same (37.64%)  
*Use "report_statistics -detailed_analysis" for details.  
-----  
Coverage  
-----  
    test_coverage          29.07% 29.61%  
    fault_coverage        24.20% 24.58%  
    atpg_effectiveness   100.00% 100.00%  
-----  
#test_patterns          4  
  #clock_sequential_patterns  4  
#simulated_patterns     64  
CPU_time (secs)         0.5  
-----
```

Here is how the statistics report would look if you used a “report_statistics -hierarchy -level 1” command:

Hierarchical Statistics Report Stuck-at Faults						
Instance	Module	#faults	UO	AU	Test	Coverage
-top-	CHPJ	45084	0	14465	59.12%	
CHPJ_MGsg_FLP_DCDR_INST	CHPJ_MGsg_FLP_DCDR	34	0	7	79.41%	
DEF	CHPJ_LV_BGROUP_DEF	1638	0	166	89.85%	
LVSN_JTAP_INST	CHPJ_LVISION_JTAP	0	0	0	0.00%	
LVSN_LGCTST_INSTCHP	LVSN_LGCTST	0	0	0	0.00%	
LV_BRST_CLK_CTRL_I1	CHPJ_LV_BRST_CLK_CTRL_BL5_P_LVU67	0	0	0	0.00%	
LV_BRST_CLK_CTRL_I2	CHPJ_LV_BRST_CLK_CTRL_BL5_P	0	0	0	0.00%	
LV_SE_CTRL_BL5_P_I1	CHPJ_LV_SE_CTRL_BL5_P_LVU69	356	0	29	77.34%	
LV_SE_CTRL_BL5_P_I2	CHPJ_LV_SE_CTRL_BL5_P	356	0	19	76.83%	
LV_SHIFT_CLK_CTRL_INST	CHPJ_LV_SHIFT_CLK_CTRL	0	0	0	0.00%	
TLB_inst	TLB	42512	0	14100	57.69%	

Example 2

The following example shows a statistics report with detailed analysis:

report_statistics -detailed_analysis

```
Statistics Report
Stuck-at Faults
-----
Fault Classes          #faults
                      (total)
-----
FU (full)           144
-----
UU (unused)         14 ( 9.72%)
RE (redundant)      2 ( 1.39%)
AU (atpg_untestable) 128 (88.89%)
-----
Fault Sub-classes
-----
AU (atpg_untestable)
  TC (tied_cells)      124 (86.11%)
  /u4/u2 (38) TX       23 (15.97%)
  /u4/u5/u2 (40) TX    18 (12.50%)
  /u5/u2 (42) TX       18 (12.50%)
  /u2 (36) TX          14 ( 9.72%)
  /u4/u5/u3 (41) TX    14 ( 9.72%)
  /u7 (37) TX          10 ( 6.94%)
  /u4/u3 (39) TX       10 ( 6.94%)
  (Combined tied cells) 17 (11.81%)
  Unclassified         4 ( 2.78%)
-----
Coverage
-----
test_coverage        0.00%
fault_coverage       0.00%
atpg_effectiveness  100.00%
-----
#test_patterns        0
#simulated_patterns   0
CPU_time (secs)      1.8
-----
```

Use the -Threshold switch if you want to limit the detailed list of untested faults. Use the following command to show the detailed breakout of the DI fault sub-classes in addition to the AU and UD sub-classes:

```
report_statistics -detailed_analysis DI
```

```
Statistics Report
Stuck-at Faults
-----
...
-----
Fault Sub-classes
-----
DI (det_implication)          30 ( 0.06%)
EDT (edt_logic)              600 ( 1.20%)
SCAN (scan_path)              2146 ( 4.30%)
SEN (scan_enable)             180 ( 0.36%)
CLK (clock)                   300 ( 0.60%)
SR (set_reset)                30 ( 0.06%)
MEM (memory)                  30 ( 0.06%)
DIN (data_in)                 30 ( 0.06%)
...
```

Example 3

The following example shows the statistics report for relevant coverage. A second column is added to the report to compare statistics. The excluded fault sub-classes are marked as “deleted”.

```
set_relevant_coverage -exclude fp
report_statistics
```

Statistics Report Stuck-at Faults		
Fault Classes	#faults (total)	#faults (total relevant)
FU (full)	8078584	8048951
UC (uncontrolled)	2079 (0.03%)	same (0.03%)
UO (unobserved)	92608 (1.15%)	same (1.16%)
DS (det_simulation)	4797074 (59.38%)	same (59.39%)
DI (det_implementation)	677274 (8.38%)	same (8.39%)
PU (posdet_untestable)	245 (0.00%)	same (0.00%)
PT (posdet_testable)	4 (0.00%)	same (0.00%)
UU (unused)	21254 (0.26%)	same (0.27%)
TI (tied)	602348 (7.46%)	same (7.47%)
BL (blocked)	428144 (5.30%)	same (5.31%)
RE (redundant)	12063 (0.15%)	same (0.15%)
AU (atpg_untestable)	1445491 (17.89%)	1415858 (17.59%)
<hr/>		
Fault Sub-classes		
AU (atpg_untestable)		
BB (black_boxes)	148646 (1.84%)	same (1.90%)
FP (false_paths)	165611 (2.05%)	deleted
Unclassified	852523 (10.55%)	same (10.59%)
UC+UO		
AAB (ATPG_abort)	88056 (1.09%)	same (1.09%)
Unclassified	6631 (0.08%)	same (0.08%)
<hr/>		
Coverage		
test_coverage	78.04%	79.67%
fault_coverage	67.77%	68.37%
atpg_effectiveness	98.83%	98.83%
<hr/>		
#test_patterns		17206
#clock_sequential_patterns		17206
#simulated_patterns		18880
CPU_time (secs)		5579.7
<hr/>		

Example 4

The following example shows the Clock Domain Summary excerpted from a statistics report:

report_statistics -clock_domains summary

Statistics Report
Stuck-at Faults

...

Clock Domain Summary	% faults (total)	Test Coverage (total relevant)
/pll_ref_clk	10.06%	8.19%
/pll_ref_clk	20.00%	0.00%
/tbcin	3.03%	32.76%
/reset_tbc20	0.01%	6.25%
/pt_clk	1.02%	78.73%
/refclk	0.09%	37.57%
/reset_rbc	0.01%	6.67%
/func1	40.22%	25.25%
/func2	18.76%	13.18%
/klok1	0.09%	24.43%
/klok2	0.00%	0.00%
/klok3	0.00%	0.00%
/klok4	0.14%	27.20%
/RESET_ALL	0.99%	0.21%
/arm_clk2	0.02%	38.04%
/arm_clk3	27.61%	17.95%
/ddr_clk1x	2.57%	35.08%
/ddr_reset	0.27%	0.92%
/LSI_TCK	2.75%	19.28%
/in_hcm[12]	0.13%	0.33%
/in_hcm[10]	0.00%	0.00%
/in_hcm[7]	0.18%	5.07%
/in_hcm[6]	2.19%	1.29%
/in_hcm[5]	0.74%	2.06%
/in_hcm[4]	18.76%	13.18%
/in_hcm[3]	0.14%	27.20%
/in_hcm[2]	0.00%	0.00%
/in_hcm[1]	0.96%	21.75%
/in_hcm[0]	0.43%	21.80%
/bidi[15]	0.45%	21.29%
/bidi[14]	0.04%	2.00%

Example 5

The following example performs timing-aware ATPG using SDF timing data. Note that the statistics report includes delay test coverage:

```
set_system_mode analysis
read_sdf my_design.sdf
set_atpg_timing on -clock_waveform DEFAULT 3 1 1
create_patterns
report_statistics
```

Statistics Report		
Transition Faults		
Fault Classes	#faults (coll.)	#faults (total)
FU (full)	3370612	4741152
UC (uncontrolled)	846	1339
UO (unobserved)	41993	63027
DS (det_simulation)	2518486	3799766
DI (det_implication)	494643	516429
PU (posdet_untestable)	34	52
UU (unused)	30	52
TI (tied)	4270	5308
BL (blocked)	2654	2754
AU (atpg_untestable)	307656	352425
test_coverage	89.58%	91.19%
fault_coverage	89.39%	91.04%
atpg_effectiveness	98.73%	98.64%
delay_test_coverage	68.09%	67.39%
#test_patterns		3508
#clock_sequential_patterns		3508
#simulated_patterns		3616
CPU_time (secs)		17372.7

Example 6

The following example shows the multiple detection test coverage report at the end of the normal report_statistics output when some unlisted faults are loaded using the “read_faults -graybox” command and AU.PC faults are excluded from the relevant test coverage.

```
read_faults top_faults.txt -retain
read_faults core1_faults.txt -graybox -retain
set_relevant_coverage -exclude au.pc
report_statistics
```

The following shows the left side of the command output:

```
Statistics Report
Stuck-at Faults
-----
...
Multiple Detection Statistics
-----
Detections          DS Faults
(N)                (Detections == N) (total w. unlisted)
-----
1                  0 ( 0.00%)    15563 ( 17.22%)
2                  0 ( 0.00%)    15563 ( 17.22%)
...
8                  135 ( 0.11%)   15249 ( 16.89%)
9                  198 ( 0.16%)   15127 ( 16.76%)
10+                5238 ( 4.12%)  14950 ( 16.57%)
-----
bridge_coverage_estimate
-----
```

The following shows the right side of the command output, with the Detections column included for your reference.

```
Multiple Detection Statistics
-----
Detections          Test Coverage (Detections >= N)
(N)                (relevant w. unlisted) (total w/o unlisted) (relevant w/o unlisted)
-----
1                  17.30%      18220 ( 12.16%)     12.17%
2                  17.30%      18220 ( 12.16%)     12.17%
...
8                  16.97%      17870 ( 11.76%)     11.78%
9                  16.84%      17735 ( 11.61%)     11.62%
10+                16.65%      17537 ( 11.39%)     11.40%
-----
bridge_coverage_estimate
-----
```

Example 7

The following example shows the statistical fault report when gray or black boxes are present:

report_statistics -Hierarchy -LEVel 2

Hierarchical Statistics Report Stuck-at Faults						
Instance	Module	#faults	UO	AU	Test	Coverage
-top-	tde_top_bscan	1602	2316	768	93.16%	
bsr_il	bsr_instance_1	0	0	0	0%	
coria	tde_top	1466	36	95	90.97%	
u1(GB)	td_edt	20	0	10	0.00%	
u2	t_des	1466	36	95	90.79%	
pad_il	pad_inst1	10	0	10	0.00%	
tdbuf (BB)	tri_en_hi	10	0	10	0.00%	
tap_i	tap	0	0	0	0.00%	

Example 8

The following example shows that the rows for “Instances Not in netlist” and “Instances in netlist” display, because the command is set to the level of the gray box. In such a case, the report also shows the next lower level instances for the gray box:

report_statistics -hierarchy -LEVel 3

Hierarchical Statistics Report Stuck-at Faults						
Instance	Module	#faults	UO	AU	Test	Coverage
-top-	tde_top_bscan	1602	2316	768	93.16%	
bsr_il	bsr_instance_1	0	0	0	0%	
coria	tde_top	1466	36	95	90.97%	
u1(GB)	td_edt	20	0	10	0.00%	
Instances Not in netlist		15	0	5	0.00%	
Instances in netlist		5	0	5	0.00%	
inst1	xxxx	3	2	1	0.00%	
inst2	yyyy	2	0	10	0.00%	
u2	t_des	1466	36	95	90.79%	
pad_il	pad_inst1	10	0	10	0.00%	
tdbuf (BB)	tri_en_hi	10	0	10	0.00%	
tap_i	tap	0	0	0	0.00%	

Example 9

The following example shows the hierarchical statistics report using the **-grayboxes_only** switch:

report_statistics -hierarchy -grayboxes_only

Hierarchical Statistics Report (Grayboxes Only) Stuck-at Faults						
Instance	Module	#faults	UO	AU	Test	Coverage
/elt_inst (GB)	elt	18198	8	242	98.44%	
Instances not in netlist	elt	15176	8	88	99.30%	
Instances in netlist	elt	3022	0	154	93.51%	

Example 10

The following example shows the hierarchical statistics report using the -EXClude_unlisted_faults switch:

```
report_statistics -Hierarchy -Level 3 -EXClude_unlisted_faults
```

Hierarchical Statistics Report Stuck-at Faults						
Instance	Module	#faults	UO	AU	Test	Coverage
-top-	tde_top_bscan	1602	2316	768	93.16%	
bsr_il	bsr_instance_1	0	0	0	0	0%
coria	tde_top	1466	36	95	90.97%	
u1 (GB)	td_edt	20	0	10	0.00%	
inst1	xxxx	3	2	1	0.00%	
inst2	YYYY	2	0	10	0.00%	
u2	t_des	1466	36	95	90.79%	
pad_il	pad_inst1	10	0	10	0.00%	
tdbuf (BB)	tri_en_hi	10	0	10	0.00%	
tap_i	tap	0	0	0	0.00%	

Example 11

This is an example of the hierarchical statistics report using -detection option to adjust the report to reflect the values with detections greater than or equal to two.

```
report_statistics -hierarchy -detections 2 -level 1 -exclude_unlisted_faults -format "DS %tc  
%rtc %fc %ae"
```

Hierarchical Statistics Report Stuck-at Faults (Detections >= 2)						
Instance	DS	Test Coverage	TC Loss	Relevant TC	Fault Coverage	ATPG Eff.
-top-	13324	59.76%	40.24%	56.76%	47.44%	99.28%
CHPJ_Mg_I_CD_FLP_DCDR_I	21	79.41%	0.02%	79.41%	79.41%	100.00%
DEF	504	89.85%	0.49%	89.80%	89.74%	100.00%
LVISION_JTAP_INST	0	0.00%	0.00%	0.00%	0.00%	0.00%
LVISION_LOGICTEST_INST	0	0.00%	0.00%	0.00%	0.00%	0.00%
LV_BURST_CLK_CTRL_I1	0	0.00%	0.00%	0.00%	0.00%	0.00%
LV_BURST_CLK_CTRL_I2	0	0.00%	0.00%	0.00%	0.00%	0.00%
LV_SE_CTRL_BL5_P_I1	85	77.34%	0.09%	76.74%	27.81%	100.00%
LV_SE_CTRL_BL5_P_I2	55	76.83%	0.06%	76.83%	17.70%	100.00%
LV_SHIFT_CLK_CTRL_INST	0	0.00%	0.00%	0.00%	0.00%	0.00%
TLB_inst	12636	58.29%	39.15%	55.18%	46.23%	99.24%

Example 12

This is an example of the hierarchical statistics report using -detections option to adjust the report to reflect the values with detections greater than or equal to five and limited with the -grayboxes_only option.

```
report_statistics -hierarchy -detections 5 -grayboxes_only -format "DS %tc tcl %rtc %fc %ae"
```

Hierarchical Statistics Report (Grayboxes Only)							
Stuck-at Faults (Detections >= 5)							
Instance	DS	Test Coverage	TC Loss	Relevant TC	Fault Coverage	ATPG Eff.	
/T_inst/E_WPR_IST (GB)	685	25.19%	15.78%	20.29%	13.77%	94.86%	
Instances not in netlist	0	27.77%	3.87%	27.60%	18.69%	88.28%	
Instances in netlist	685	24.31%	11.91%	18.39%	12.49%	96.57%	

Example 13

The following example shows AU fault subclasses:

Statistics Report		
Stuck-at Faults		
Fault Classes	#faults (total)	#faults (total relevant)
FU (full)	40550	29673
UO (unobserved)	3 (0.01%)	same (0.01%)
DS (det_simulation)	14108 (34.79%)	same (47.54%)
DI (det_implication)	5111 (12.60%)	same (17.22%)
PU (posdet_untestable)	11 (0.03%)	same (0.04%)
UU (unused)	2416 (5.96%)	same (8.14%)
TI (tied)	24 (0.06%)	same (0.08%)
BL (blocked)	7 (0.02%)	same (0.02%)
RE (redundant)	284 (0.70%)	same (0.96%)
AU (atpg_untestable)	18586 (45.83%)	7709 (25.98%)
Fault Sub-classes		
AU (atpg_untestable)		
EDT (edt_blocks)	603 (1.49%)	deleted
PC* (pin_constraints)	15 (0.04%)	same (0.05%)
TC* (tied_cells)	7642 (18.85%)	same (25.75%)
OCC (on_chip_clock_control)	454 (1.12%)	deleted
IJTAG (ijtag)	873 (2.15%)	deleted
LPCT (low_pin_count_test)	8947 (22.06%)	deleted
Unclassified	52 (0.13%)	same (0.18%)
UC+UO		
UNS (unsuccess)	3 (0.01%)	same (0.01%)
*Use "report_statistics -detailed_analysis" for details.		
Coverage		
test_coverage	50.83%	71.36%
fault_coverage	47.41%	64.79%

Related Topics

[set_atpg_timing](#)

[set_relevant_coverage](#)

report_synchronous_clock_groups

Context: dft, patterns

Mode: setup, analysis

Displays a list of all user-defined synchronous clock groups.

Usage

`report_synchronous_clock_groups`

Note

 Synchronous clock groups have no impact on scan chain analysis, test point identification, and EDT IP insertion (in the dft -scan, dft -test_points and dft -edt sub-contexts).

Description

Displays a list of all user-defined synchronous clock groups.

The `report_synchronous_clock_groups` command displays a list of all clock groups that you defined with the `add_synchronous_clock_group` command. For more information about synchronous clock groups, refer to the [add_synchronous_clock_group](#) command description.

Arguments

None.

Examples

The following example creates two clock groups and lists them out.

```
SETUP> add_synchronous_clock_group {I1 I2}
SETUP> report_synchronous_clock_groups
```

```
User-defined Synchronous Clock Group
=====
// No.  Clocks (Labels)
//   --- -----
//     1    I1
//       I2
```

```
SETUP> add_synchronous_clock_group {I3 I4}
SETUP> report_synchronous_clock_groups
```

```
User-defined Synchronous Clock Groups
=====
// No.  Clocks (Labels)
//   --- -----
//     1    I1
//       I2
//     2    I3
//       I4
```

Related Topics

[add_clocks](#)
[add_synchronous_clock_group](#)
[analyze_control_signals](#)
[delete_clocks](#)
[delete_synchronous_clock_group](#)

report_tcd_ldb_validation

Context: patterns -scan_diagnosis

Mode: setup

Verifies that the core instance pathnames and hierarchy in the TCD matches the core instance pathnames and hierarchy in the LDB.

Usage

```
report_tcd_ldb_validation -tcd tcdfile ... -ldb ldb_dir ...
```

Arguments

- **-tcd tcdfile ...**
Required switch and string pair that specifies one or more TCD files.
- **-ldb ldb_dir ...**
Required switch and string pair that specifies one or more LDB directories.

Description

The command extracts and compares the set of chip designs and core instance paths from the TCD files and the LDBs. The report lists the coverage of each instance path in the TCD files and the LDBs. The tool issues an error when the names do not match.

The typical flow runs ATPG, LDB creation, and diagnosis in separate phases. Run report_tcd_ldb_validation after ATPG and LDB creation to ensure that the correct data will available at diagnosis. Refer to “[Diagnosis for Hierarchical Design](#)” in the *Tessent Diagnosis User’s Manual* for details.

Examples

The following example validates the chip.tcd against the core data within the core.lbd and chip.lbd directories.

```
report_tcd_ldb_validation -tcd chip.tcd -ldblist core.lbd
```

Core validation report:

TCD Files:

Index	Filename	Design
1>	chip.tcd	chip_top_edt

LDB Directories:

Index	Directory	Design
1>	core.ldb	uUARTs_i

Instance Coverage:

Instance	TCDs	LDBs
cpu/core_1	1	1

The following example shows the output when you have multiple TCDs and cores. In the transcript, the third and fourth TCD files refer to the same chip with different pattern sets. Each chip has embedded cores and a chip level LDB. The first LDB refers to core “core_D,” which is not found in the LDB directories. The second LDB contains a top-level chip, which is not found in any TCD file leading to the chip’s instance not being found.

```
set tcd { chip_1.tcd chip_2.tcd chip_3.tcd }
set ldb { core_A.ldb core_B.ldb core_C.ldb chip_1.ldb chip_2.ldb chip_3.ldb chip_4.ldb }
report_tcd_ldb_validation -tcd $tcclist -ldb $ldblist
```

Core validation report:

TCD files:

Index	Filename	Design
1	chip_1.tcd	chip_1_edt_top
2	chip_2.tcd	chip_2_edt_top
3	chip_3.tcd	chip_3_edt_top
4	chip_4.tcd	chip_3_edt_top

Index	Directory	Design
1	core_A.ldb	core_A
2	core_B.ldb	core_B
3	core_C.ldb	core_C

Instance	TCDs	LDBs
cpu/core_A_1	1	1
cpu/core_A_2	1	1
cpu/core_B_1	1	2
cpu/core_D_1	1	[Not Found]
cpu/core_B_1	2	2
cpu/core_C_1	2	3
cpu/core_C_1	3 4	3
chip_4/core_B_1	[Not Found]	2

report_tcl_shell_options

Context: all contexts

Mode: setup

Lists the currently specified Tcl shell options.

Usage

`report_tcl_shell_options`

Description

Lists the currently specified Tcl shell options as specified with the [set_tcl_shell_options](#) command.

Arguments

None

Related Topics

[get_tcl_shell_option](#)

[set_tcl_shell_options](#)

report_test_end_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Reports a single view of all existing iCalls specified with the [set_test_end_icall](#) command.

Usage

`report_test_end_icall`

Description

Reports a single view of all existing iCalls that were specified with the [set_test_setup_icall](#) command.

Arguments

None.

Related Topics

[report_test_setup_icall](#)

[set_test_setup_icall](#)

report_test_logic

Context: dft -scan, dft -test_points

Mode: insertion

Prerequisites: Test logic or test points must be added with the [insert_test_logic](#) command.

Displays the test logic that the tool added during the scan insertion process.

Usage

```
report_test_logic [-Instance | -Module | -Summary] [<> | >>] file_pathname
```

Description

Displays the test logic that the tool added during the scan insertion process.

The report_test_logic command displays information about the additional library cells instantiated in the design during the scan insertion process.

Arguments

- -Instance

An optional switch that displays the list of instance pathnames and the corresponding DFT library models inserted by the tool. This option also includes a summary of the number of each instance-based DFT library model that the tool inserted. This is the default.

- -Module

An optional switch that displays the list of module names, the list of instance pathnames, and the corresponding DFT library models that the tool inserted as test logic and test points. This option also includes a summary of the number of each module-based DFT library model that the tool inserted.

- -Summary

An optional switch that displays a summary that contains the number of each module and instance-based DFT library model that the tool inserted.

- > file_pathname

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> file_pathname

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Related Topics

[insert_test_logic](#)

[report_test_points](#)

report_test_points

Context: dft -test_points

Mode: all modes

Displays test points that are inserted with the [insert_test_logic](#) command.

Usage

```
report_test_points [object_spec] [-columns column_names]  
[-add_columns column_names]
```

Description

Use this command to report the test points that are inserted with the [insert_test_logic](#) command.

The report is printed in a table format where each column represents the test point attributes, such as name, location, type, or internal_clock_domain, for each test_point. Use the -columns switch to specify which columns to show and the order in which they appear. By default, report_test_points will print the following test_point attributes:

```
name, type, control_point_type, enable_connection_name, clock_domain,  
flop_location.
```

You can add additional columns after the default columns (or after the columns specified with the –columns switch) using the -add_columns switch.

The following table summarize what is new or changed in the report_test_points command and provides a mapping between the old and new command options.

Previous command option	New command option
-internal_clock_paths	-add_columns clock_connection_name
-sharing id	-add_columns shared_id
Summary	(No longer supported)
-control/-observe	report_test_points [get_test_points –type control/ observe]
-tool_generated_only	report_test_points [get_test_points –origin design_analysis]

Arguments

- *object_spec*

An optional value that specifies a Tcl list of one or more object names or a collection of one or more test_point objects.

- **-columns *column_names***

An optional value that specifies a Tcl list of one or more object names that are the attributes of a test_point object. See “[Test Point Data Model](#)” for the list of Test point attributes that are valid column names.

- **-add_columns *column_names***

An optional list of columns to append after the columns specified by the -columns switch (or after the default columns). See “[Test Point Data Model](#)” for the list of Test point attributes that are valid column names.

Examples

Example 1

The following example shows the default columns printed when the command is issued with no arguments:

```
report_test_points
```

```
=====
name          type   control_point_type enable_connection_name  clock_domain
flop_location
-----
tp#test_module2/neg1/Z    control  AND           /lbist_en        clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_i

tp#test_module2/neg2/Z    control  OR           /lbist_en        clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i

tp#1/test_module2/neg1/Z  observe  --          /lbist_en        clk2
/test_module2/ts_default_PD_u12_op_120sffp1_i

tp#1/test_module2/neg2/Z  observe  --          /lbist_en        clk2
/test_module2/ts_default_PD_u12_op_122sffp1_i

tp#test_module1/neg1/Z    control  AND           /lbist_en        clk2
/test_module1/ts_default_PD_u11_cp_0sffp1_i

tp#test_module1/neg2/Z    control  OR           /lbist_en        clk2
/test_module1/ts_default_PD_u11_cp_4sffp1_i

tp#1/test_module1/neg1/Z  observe  --          /lbist_en        clk2
/test_module1/ts_default_PD_u11_op_112sffp1_i

tp#1/test_module1/neg2/Z  observe  --          /lbist_en        clk2
/test_module1/ts_default_PD_u11_op_114sffp1_i

tp#u1/U15/Z              observe  --          /lbist_en        clk1
/u1/ts_default_PD_default1_op_128sffp1_i

tp#u1/U14/Z              observe  --          /lbist_en        clk1
/u1/ts_default_PD_default1_op_126sffp1_i

tp#u1/U75/Z              observe  --          /lbist_en        clk2
/u1/ts_default_PD_default1_op_212sffp1_i

tp#U64/Z                observe  --          /lbist_en        clk2
/ts_default_PD_default1_op_52sffp1_i

tp#u1/U71/Z              observe  --          /lbist_en        clk2
/u1/ts_default_PD_default1_op_206sffp1_i

...

```

Example 2

The following example shows only control points below the test_module2 instance:

```
report_test_points [get_test_points -below_instances test_module2 -type control]
```

```
=====
name          type   control_point_type  enable_connection_name  clock_domain
flop_location
-----
tp#test_module2/neg1/z  control  AND           /lbist_en           clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_itp#test_module2/neg2/z  control  OR
/lbist_en             clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i
```

Related Topics

[delete_test_points](#)
[insert_test_logic](#)
[report_test_logic](#)
[add_input_constraints](#)
[set_test_point_insertion_options](#)

report_test_point_statistics

Context: dft -test_points

Mode: analysis (after analyze_test_points is run), insertion

Use this command to report statistics regarding the location of test points inside the design.

Usage

```
report_test_point_statistics [object_spec] [-top lines] [-columns column_names]  
[-sort {column ...}] [-add_columns column_names]
```

Description

The report_test_point_statistics command provides statistical information in a tabular format regarding the dispersion of test points throughout the design. This can help identify parts of the design that can be modified for greater controllability and/or observability and to reduce the number of test points needed.

By default, when no arguments or switches are specified, the output includes a row for each instance in the design that includes at least one test point. Each row will contain the module name (if the name is not unique it will include an asterisk followed by a unique index), columns that include the number of test points, control points and observe points in the specified instance (including test points in submodules below the module), and the full path to the instance.

The lines will be sorted in descending order according to the number of test points.

Note

 The command is only available after test point analysis and insertion of test points.

Arguments

- *object_spec*
An optional value that specifies a Tcl list of one or more modules or instances in the design, that controls which instances will be reported.
- *-top lines*
An optional switch and integer pair that specifies the number of the top rows of the report to be printed, not including the header.
- *-columns {*columns* ... }*
An optional switch that specifies to organize the output format according to a specified list of columns. The allowed columns are:
 - module
 - test_points
 - control_points

- observe_points
- instance

This switch also allows you to change the order of the columns and/or hide default columns.

- -sort *column*
An optional switch and column name that sorts the output rows according to the value of the specified column in descending order. Default is to sort according to the test_points column.
- -add_columns *column_names*
An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

Example 1

The following example prints the top two rows of the default report.

```
report_test_point_statistics -top 2
```

```
// command: report_test_point_statistics -top 2
=====
module          test_points  control_points  observe_points  instance
-----
MT_GET (top)    21 (100.0%)   14 (100.0%)    7 (100.0%)    /
MT_SET_DATA_WIDTH32 19 (90.5%)   14 (100.0%)    5 (71.4%)    /MT_BLK
```

Example 2

The following example reports the test point statistics for the specified instances.

```
report_test_point_statistics {MT_SET_DATA_WIDTH32 LCG_DATA_WIDTH32_1}
```

```
// command: report_test_point_statistics {MT_SET_DATA_WIDTH32 LCG_DATA_WIDTH32_1}
=====
module          test_points  control_points  observe_points  instance
-----
MT_GET (top)    21 (100.0%)   14 (100.0%)    7 (100.0%)    /
MT_SET_DATA_WIDTH32 19 (90.5%)   14 (100.0%)    5 (71.4%)    /MT_BLK
LCG_DATA_WIDTH32_1 3 (14.3%)    3 (21.4%)     0 (0.0%)     /MT_BLK/LCG2
```

report_test_setup_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Reports a single view of all existing iCalls specified with the set_test_setup_icall command.

Usage

```
report_test_setup_icall
```

Description

Reports a single view of all existing iCalls specified with the set_test_setup_icall command.

Each iCall is reported as follows:

```
iCall iproc_name [arguments ...]
```

When multiple iCalls are merged, the command reports these iCalls within the “iMerge –begin” and “iMerge –end” as follows:

```
iMerge -begin  
  iCall iproc_name [arguments ...]  
  ...  
iMerge -end
```

Arguments

None.

Related Topics

[report_test_end_icall](#)
[set_test_setup_icall](#)

report_test_stimulus

Context: dft -edt, patterns -scan

Mode: analysis

Displays the stimulus necessary to satisfy the specified set, write, or read conditions.

Usage

```
report_test_stimulus
  -Set {{id# | pin_path_name} {0 | 1 | Z}}...
  |-Cycle_set {[F]Irst_events | [L]Eading_edge_events | [LAst_events]
    {{id# | pin_path_name} {0 | 1 | Z} cycle#}...}...
  |-Write {{id# | instance_name} address_values [data_values]}...
  |-Read {{id# | instance_name} address_values}...
  |-RWx {{id# | instance_name} address_values}...
  |-SENsitize {{id# | instance_name | pin_path_name}
    [-Observe_at {id# | instance_name | pin_path_name} [-Expect {0 | 1 | X | Z}]]}
  |-Port port_number
  [-Verb[ose] | -Noverbose] [-PRevious] [-STore] [> | >>] file_pathname]
```

Description

Displays the stimulus necessary to satisfy the specified set, write, or read conditions.

This command identifies how to sensitize scan chain blockage points. For example, if you first delete all scan groups and then go to the ATPG system mode, you can issue the report_test_stimulus command for possible conditions to satisfy sensitization. That is, if the blockage was at an AND gate, you can try to set an input of the gate to 1.

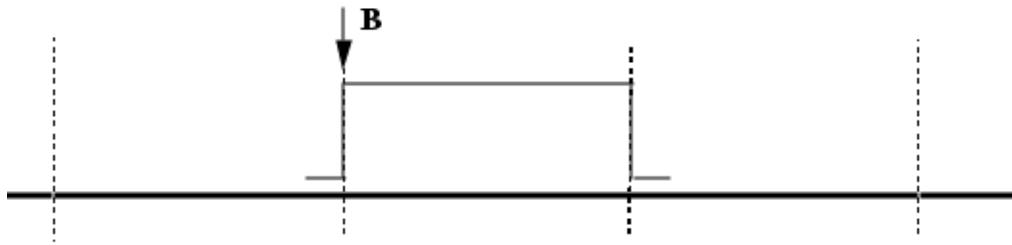
If test generation is successful, the stimulus necessary to satisfy the specified conditions is displayed. The stimulus for scan cells is identified by the gate index number, instance name, and cell ID number of the scan chain.

You can access the simulated values for internal gates by using the set_gate_report command with a parallel_pattern of 0.

You can also use the “report_test_stimulus -Cycle_set” command to specify the conditions at different event times or test cycles for analysis (for example, sequential controllability analysis and transition fault analysis). Each test cycle consists of at most three sets of events (clock-off, leading edge, and trailing edge) simulated in a single cycle. The following is a description of what determines the set of events within which the justified/simulated event occurs:

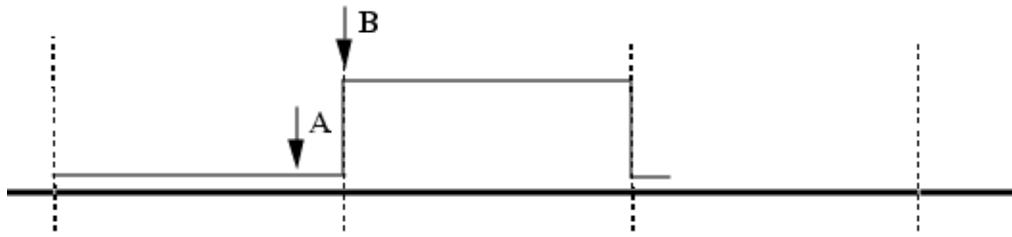
- When both “clock-off simulation” and “split capture” are turned off (the default unless the “set_clock_off_simulation On” and “set_split_capture_cycle On” commands are issued or set by ATPG Expert), a cycle includes one set of simulation events; that is, there are no clock-off events and the leading edge and the trailing edge are handled in one time frame. The simulation occurs at point B as shown in [Figure 5-11](#).

Figure 5-11. set_clock_off_simulation Off and set_split_capture_cycle Off



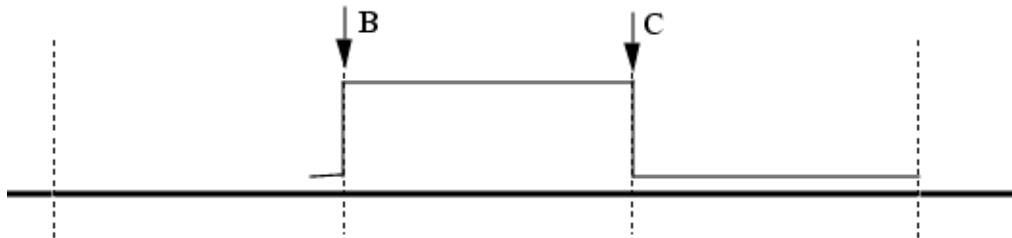
- When “clock-off simulation” is turned on and “split capture” is turned off, a cycle includes two sets of simulation events: clock-off events, and the events handling both the leading edge and the trailing edge events. The two simulation points for this case are at points A for clock-off events and B for leading and trailing edge events with “split capture” off as shown in [Figure 5-12](#).

Figure 5-12. set_clock_off_simulation On and set_split_capture_cycle Off



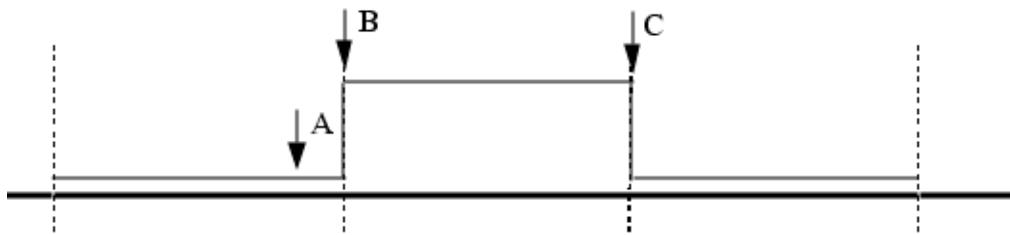
- When “clock-off simulation” is turned off and “split capture” is turned on, a cycle includes two sets of simulation events: the leading edge and trailing edge events. For this case (as shown in [Figure 5-13](#)), the simulation points are point B for the leading edge events and point C for the trailing edge events. Turning on “split capture” is necessary if a leading edge flip-flop feeds a trailing edge flip-flop in the functional path.

Figure 5-13. set_clock_off_simulation Off and set_split_capture_cycle On



- When both “clock-off simulation and” “split capture” are turned on, a cycle includes three sets of events: clock-off, the leading edge, and the trailing edge. [Figure 5-14](#) includes the three simulation points: A for clock-off, B for leading edge, and C for trailing edge events.

Figure 5-14. set_clock_off_simulation On and Set Split Capture On



Arguments

- **-Set *id# | pin_path_name* 0 | 1 | Z**

A switch with a repeatable argument and literal pair that specifies the pin and its value for which you want to generate the appropriate stimulus. You may specify multiple argument pairs with a single -Set switch.

id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate cannot be a RAM or ROM gate.

pin_path_name — A string that specifies the pathname of the pin you want to set.

0 — A literal that sets the *id#* or *pin_path_name* to 0.

1 — A literal that sets the *id#* or *pin_path_name* to 1.

Z — A literal that sets the *id#* or *pin_path_name* to Z.

- **-Cycle_set {[-FIrst_events | -LEading_edge_events | -LAst_events] { {*id# | pin_path_name*} {0 | 1 | Z} *cycle#*} ...}**

A switch and repeatable argument, literal, and number set that specifies the pin, its value, and the cycle to set the value at the pin for which you want to generate the appropriate stimulus.

The events switches determine when the value is justified/simulated in the cycle. The events switches hold for all triples (pin, value, cycle) specified thereafter until another events switch is specified in the same command. For example, you can specify “report_test_stimulus -first <triple> <triple> -last <triple>” rather than “report_test_stimulus -first <triple> -first <triple> -last <triple>”.

-FIrst_events — A switch that sets the value of all pin triplets following this switch when the first events are simulated. If the “set_clock_off_simulation On” command has been issued or set by ATPG Expert, this event is the clock-off simulation; otherwise, it is always the present leading edge event simulation.

-LEading_edge_events — A switch that sets the value of all pin triplets following this switch at the leading edge event simulation. ATPG always has one set of events, called the “leading edge,” that it simulates by default in all cycles. Leading edge always refers to that one possibility.

-LAst_events — A switch that sets the value of all pin triplets following this switch in the last set of simulation events that occur in every cycle. This is the default. If the “set_split_capture_cycle On” command was issued or set by ATPG expert, this event

is the trailing edge events simulation. By default, the value of all pins is set at the trailing edge event in the cycle if an events switch is not specified in the -Cycle_set switch. Also, when there are several -Cycle_set switches specified in the same command line, each -Cycle_set switch resets its default time frame to the trailing edge frame regardless of the time frame specified in the previous -Cycle_set switch.

id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate *id#* cannot be a flattened RAM or ROM gate.

pin_path_name — A string that specifies the pathname of the pin you want to set.

0 — A literal that sets the *id#* or *pin_path_name* to 0.

1 — A literal that sets the *id#* or *pin_path_name* to 1.

Z — A literal that sets the *id#* or *pin_path_name* to Z.

cycle# — An integer that specifies the post-load capture cycle in which the event is to occur.

With named capture procedures, the first cycle after the first scan load is “1”, the second cycle after first scan load is “2”, and so on.

Without named capture procedures, “1” indicates the cycle may be anywhere in the window and can “slide” within the allocated depth. The *cycle#* earlier is always “0” if it exists, before that “-1” if it exists, and so on. The *cycle#* after “1” is always “2”, and so on. For example, if the depth is set to 1 by you or ATPG Expert, then there is only cycle “1”. If the depth is set to 2, the cycles could be numbered “0” then “1” or “1” then “2”. In this case, if you specified “0” and “1” for *cycle#*, the cycles are the first and second cycles and there is no cycle “2”. However, if you specified “1” and “2” for *cycle#*, the cycles are the first and second cycles, and there is no cycle “0”.

Note

 When used without named capture procedures, -Cycle_set is not constrained from satisfying control requirements in the observation cycle(s). Therefore, it is possible for it to succeed although ATPG (stuck-fault, transition fault, MacroTest read/observe pattern, etc.) would fail, because they typically will have reserved at least one observation cycle. Such false successes can be prevented by temporarily reducing depth to artificially reserve observation cycle(s), or by scheduling a clock pulse or some other event in each observation cycle. Each solves the false success problem by creating -cycle_set compatibility with the cycle requirements imposed upon ATPG for it to succeed. If -Cycle_set fails, ATPG should also.

- **-Write *id# | instance_name address_values data_values***

A switch with a repeatable argument that specifies the RAM to which you want to write and, optionally, its address and data values. You may specify multiple argument triplets with a single -Write switch. The switch arguments are:

id# — An integer that specifies the identification number of the RAM gate to which you want to write.

instance_name — A string that specifies the pathname of the RAM instance to which you want to write.

address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

data_values — An optional character string consisting of 0's, 1's, and X's that specifies the values you want to place on the RAM data lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM data lines available.

If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-Read *id# | instance_name address_values***

A switch with a repeatable argument that specifies the RAM from which you want to read and, optionally, its address value. You may specify multiple argument pairs with a single -Read switch.

id# — An integer that specifies the identification number of the RAM gate from which you want to read.

instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-RWx *id# | instance_name address_values***

A switch that allows a port read to be combined with a port write by combining -write and a -previous option in the commands after -rwx. Both the read and write enables of the RAM are then asserted together. The switch arguments are:

id# — An integer that specifies the identification number of the RAM gate from which you want to read.

instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

- **-SENSitize *id# | instance_name / pin_path_name***

A switch that allows any primitive pin or pins to be targeted for sensitization; RAM pin, RAM port, ROM pin, ROM port, MUX output pin, MUX input pin, and so on. The conditions necessary to observe the output pins (or pin) of that primitive are calculated and reported.

Note

 The -SENsitize argument, like [macrotest](#), uses a functional observation technique, unlike the basic ATPG algorithm for structural faults. Consequently, there must be a single sensitizable path from the site to be observed to some observation site (scan cell or observable PO). The sensitizable path must be through combinational gates (specifically, no sequential). If the tool can sensitize this path, then ATPG should succeed. If the tool cannot sensitize this path, then ATPG will fail. This failure, however, indicates nothing about why a fault site is untestable, and you cannot make any conclusions about why ATPG is failing.

By issuing successive -sensitize options with different pin path names while also including the -previous option, you can find (and store with -store) the conditions that would allow a test to observe multiple sites simultaneously, even though they are driven by the outputs of different primitives. In this way, you can specify more than one pin for simultaneous observation. If a single primitive has multiple outputs, such as a RAM, and only the RAM instance name is given, all of the outputs of the RAM will be observed simultaneously, or a report of failure to sensitize will be issued. The switch arguments are:

id# — An integer that specifies the identification number of the RAM gate from which you want to read.

instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

pin_path_name — A string that specifies the pathname of the pin that you want to sensitize.

-Observe_at *id# | pin_path_name | instance_name*

Specifies where the sensitized primitive output should be observed (latched for comparison). The -observe_at switch only applies to the -Sensitize switch when a single pin is specified. It is possible to specify multiple -Sensitize/-Observe_at pairings by using the -Previous option. If non-interfering paths cannot be created between all simultaneously active -Sensitize/-Observe_at points, a message is issued, and the run terminated.

-Expect 0 | 1 | X | Z

The -expect modifier also only applies if a -sensitize option is given on the same command line. It is followed by the expected output value for the primitive to be sensitized (either 0, 1, or X). The X value can also be specified using lower case x, and means unspecified output value. For example, if a RAM is being sensitized, then one output value for each output bit of the RAM must be specified if output values are given using -expect. The bits should be specified in the same order as the RAM's outputs. If a RAM named mem1 is specified with outputs Dout<7>...Dout<0>, then if the expected outputs are included, there must be 8 expected values starting with the expected value for Dout<7> as the leftmost bit. For example, 1000000x after -expect could be used to specify that the RAM's outputs are expected to be 1 for Dout<7>, and 0 for all other bits except Dout<0>, the expected output of which is Unknown or Don't Care. If this RAM's outputs were sensitized to scan latches along an inverting

path, 0111111X would be stored as the expected latched results to be scanned out as the test results.

- **-Port *port_number***

An optional switch and integer pair that specifies the identification number of the port to use for reading or writing RAM. The port identification number is zero-based; that is, the first port is “port 0”. The default is 0.

- **-Verbose**

An optional switch that displays the pattern that the command creates to satisfy the specified settings. This is the default.

- **-Noverbose**

An optional switch that specifies to not display the pattern that the command creates to satisfy the specified settings.

- **-PPrevious**

An optional switch that retains the settings from the previous report_test_stimulus command and adds them to the current settings. When this switch is used, the command displays all of the retained settings. The default is to not retain the settings.

- **-STore (not available in Tesson TestKompress (EDT On))**

An optional switch that places the command-created pattern in the internal test pattern area. You can then write this pattern to a file, in any format, by using the write_patterns command. The default is to not place the pattern in the internal test pattern area.

- **> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example command line is used for determining what stimulus is needed to write 11011 to address location 01011 for a RAM gate (gate ID number is 67):

```
set_system_mode analysis
report_test_stimulus -write 67 01011 11011
```

The following is an example of the display from the previous command line:

```
// Time = 0
// Force 1 /W1 (1)
// Force 0 /A1[4] (2)
// Force 1 /A1[3] (3)
// Force 0 /A1[2] (4)
// Force 1 /A1[1] (5)
// Force 1 /A1[0] (6)
// Force 0 /OE (7)
// Force 1 /D1[0] (14)
// Force 1 /D1[1] (15)
// Force 0 /D1[2] (16)
// Force 1 /D1[3] (17)
// Force 1 /D1[4] (18)
```

Example 2

The following example command lines display the stimulus necessary to satisfy the specified conditions at different event times or test cycles. It is assumed that both “split capture” and “clock-off simulation” are turned on, and you want to determine if a slow-to-fall transition can be launched at the output pin of gate 245.

`report_test_stimulus -cycle_set 245 1 0 245 0 1`

In this example, logic 1 is set at the default/last event (trailing due to split capture) simulated in cycle 0, and logic 0 is set at the default/last event (trailing due to split capture) simulated in cycle 1.

`report_test_stimulus -cycle_set -leading_edge 245 1 0 245 0 1`

In this example, logic 1 is set at the leading edge event of cycle 0, and logic 0 is set at the leading edge event of cycle 1.

`report_test_stimulus -cycle_set -first_events 245 1 0 245 0 1`

In this example, logic 1 is set at the first event (clock-off) simulated in cycle 0, and logic 0 is set at the first event (clock-off in this example) of cycle 1.

`report_test_stimulus -cycle_set -first_events 245 1 0 -last_events 245 0 1`

In this example, logic 1 is set at the first (clock-off) event simulated in cycle 0, and logic 0 is set at the last event (trailing edge) simulated in cycle 1.

Example 3

In the following example command lines, it is assumed only split capture is turned on, and you want to determine if a slow-to-rise transition can be launched at the output pin of gate 245.

`report_test_stimulus -cycle_set 245 0 1 245 1 2`

In this example, logic 0 is set at the default/last (trailing due to split capture) event simulated in cycle 1, and logic 1 is set at the default/last (trailing due to split capture) event simulated in cycle 2.

report_test_stimulus -cycle_set -leading_edge 245 0 1 245 1 2

In this example, logic 0 is set at the leading edge event of cycle 1, and logic 1 is set at the leading edge event of cycle 2.

report_test_stimulus -cycle_set -first_events 245 0 1 245 1 2

In this example, logic 0 is set at the first (leading-edge) event simulated in cycle 1, and logic 1 is set at the first (leading-edge in this example) event of cycle 2.

report_test_stimulus -cycle_set -first_events 245 0 1 -last_events 245 1 2

In this example, logic 0 is set at the first (leading_edge) event simulated in cycle 1, and logic 1 is set at the last (trailing edge in this example) event simulated in cycle 2.

Related Topics

[write_patterns](#)

report_testbench_simulation_options

Context: unspecified, all contexts

Mode: all modes

Reports the default values for many options used in the [run_testbench_simulations](#) command.

Usage

```
report_testbench_simulation_options [-simulator {questa | vcs}]
```

Description

Reports the default values for many options used in the [run_testbench_simulations](#) command.

Arguments

- `-simulator questa | vcs`

A switch and literal that specifies the simulator you want to use for simulation. The default value is “questa” from Mentor Graphics. The other option is vcs from Synopsys.

Examples

The following example shows the result of the `report_testbench_simulation_options` command before any default value was changed and after the simulation output directory default value was changed.

```
SETUP> set_context patterns
SETUP> set_testbench_simulation_options -simulation_output_directory ./my_sim_outdir
SETUP> report_testbench_simulation_options

// Testbench simulation options
// =====
// -----
// Option           Value
// -----
// default_simulator      questa
// simulation_output_directory    ./my_sim_outdir
// keep_simulation_data      on_failure
// store_simulation_waveforms   off
// simulation_timeout        unlimited
// parallel_simulations       1
// simulator_options          {}
// waveform_configuration_commands {log -r /*; run -all; exit}
//
```

Related Topics

[check_testbench_simulations](#)
[run_testbench_simulations](#)

report_tied_signals

Context: all contexts

Mode: all modes

Displays a list of the tied floating signals and pins.

Usage

```
report_tied_signals [-Class {Full | User | System}] [{>} | >>] file_pathname]
```

Description

Displays a list of the tied floating signals and pins.

The report_tied_signals command displays either the user class, system class, or full classes of tied floating signals and pins. If you do not specify a class, the command displays all the tied floating signals and pins.

Arguments

- **-Class Full | User | System**

An optional switch and literal pair that specifies the source (or class) of the tied floating signals or pins which you want to display. The literal choices are as follows:

Full — A literal that displays all the tied floating signals or pins in the user and system class. This is the default.

User — A literal that displays only the tied floating signals or pins created using the add_tied_signals command.

System — A literal that displays only the netlist-described tied floating signals or pins.

- **>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>>file_pathname**

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays the tied floating signals from the user class:

```
add_tied_signals 1 vcc vdd
report_tied_signals -class user
```

Related Topics

[add_tied_signals](#)

[delete_tied_signals](#)

[report_black_boxes](#)
[set_tied_signals](#)

report_timeplate

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Displays the specified timeplate.

Usage

```
report_timeplate {timeplate_name | -All} [-COre core_name [-MOde mode_name]] [{> | >>}  
file_pathname]
```

Description

Displays the specified timeplate.

The report_timeplate command displays all timeplates or the specified timeplate.

Arguments

- ***timeplate_name***

A string that specifies which timeplate to display.

- **-All**

A switch that specifies for the tool to display all timeplates. This is the default.

- **-COre *core_name***

An optional switch and string pair that specifies that the tool reports the timeplate for a given core. The core must exist, or an error will be issued.

- **-MOde *mode_name***

An optional switch and string pair that specifies that the tool reports the timeplate for the given mode of a core defined by the -core *core_name* switch. The tool reports an error if this switch is specified without the -core switch. This switch is not necessary when there is only one mode for a core. If a core has more than one mode, an error will be issued if the mode is not specified.

- **> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- **>> *file_pathname***

An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the timeplate named *tp1*.

report_timeplate tp1**Example 2**

The following example shows using the -EXpand_iprocs switch to work with the set of used ports and vector callbacks in:

ANALYSIS> report_timeplate test_setup --expand_iprocs

```
procedure test_setup =
    timeplate tp1 ;
    iReset;
    // Begin expanded iCall m8051_B1_edt_i.setup edt_low_power_shift_en on ;
    // cycle 1 start at time 40
    cycle =
        annotate "+ Targets: ..."
        ...
    end;
    // End expanded iCall
    // cycle 5 starts at time 200
    cycle =
        ...
    end;
end;
```

Example 3

This is an example of using report_timeplate with the -core switch:

report_timeplate -core piccpu_maxlen16_1

```
timeplate gen_tp1 =
    force_pi 0 ;
    measure_po 100 ;
    pulse core_1_clk 200 100;
    pulse core_1_edt_clock 200 100;
    pulse core_1_ramclk 200 100;
    period 400 ;
end;
```

Related Topics

[add_scan_groups](#)
[read_procfile](#)
[report_procedures](#)
[write_patterns](#)
[write_procfile](#)

report_udfm_statistics

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Reports detailed fault coverage based on the UDFM defect models.

Usage

```
report_udfm_statistics [-INstance pathname...] [-Groups name ...]  
[-GROUPS_Only {On | Off}] [-Percent {On | Off}] [-SET_Baseline]  
[(> | >>) file_name]
```

Description

Reports detailed fault coverage based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU faults.

The report contains the following information as well as any additional columns you specify.

If no group is defined as a result of executing the command, by default every defect model is reported in a single row. The column headings are as follows:

- Type — Lists the UDFM type name that belongs to the defect model. For groups that collect defect models from multiple UDFM types, “---” is reported.
- Block — Identifies the object the defect model is defined on: Module, Cell, or Instance. For groups that collect different defect models, “---” is reported.
- Name — Lists the defect specific module, cell or instance name. For groups, the specified string is reported.
- Instances — Lists the number of instances of the specified module, cell, or instance identified in the design. This number is always greater zero because defect models that cannot be matched to an instance of the circuit are not reported.
- Port-Insts — Lists the number of instances where only port faults are active (that is, these instances have no cell internal faults). The information allows you to see if specific defects other than traditional port faults are targeted. The numbers in this column mean the following:
 - Port-Insts=0 — The UDFM definition contains fault definitions for cell internal defects. This means that not just port faults are targeted.
 - Port-Insts=Instances — When these two columns list the same number, the UDFM definition contains fault definitions for port faults only. This can be the case when fault sites have been added by using the add_fault_sites command.
 - Port-Insts > 0 and < Instances — Even if the UDFM definition contains a fault definition for cell internal defects, the cell-internal defects may have been dropped from the count because the flip-flop isn't part of a scan chain.

- Faults — Lists the number of faults that exist due to the specified defect model or group.
- Add-DT — Lists the difference between the stored baseline and the current status for detected faults (DT). This column displays only if there was a previous call of this function to store the baseline.
- GN — Lists the coverage gain between the stored baseline and the current status with $GN=(Add-DT/(DS+DI+PT+AU+UC+UO+PU)*100)$. This column displays only if there was a previous call of this function to store the baseline.

Arguments

- -INstance *pathname* ...

An optional switch and repeatable string that limits the report of UDFM statistics to one or more specific instances. The pathname argument specifies the name of a circuit block.

- -Groups *name* ...

An optional switch and repeatable string that defines a group of UDFM defect models based on the specified *name* string(s). You can specify multiple name arguments to create multiple groups. Each *name* string creates one group. The *name* string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

For example, the following string creates two groups: one with all defect models that contain an “AND” in their name and another with all defect models that contain an “OR” in their name.

```
-Groups *AND* *OR*
```

All matching models are collected regardless of whether they are a module, a cell, or an instance. Note that in the case of multiple group names, the command adds individual models to only one group, which is the first group name in the list that matches a particular model.

- -GROUPS_Only On | Off

An optional switch and literal pair that specify how UDFM defect models that do not match any of the defined groups are reported. By default, all UDFM defect models that do not match any of the defined groups are reported in a single row. If -GROUPS_Only is set to On, this row is not created.

- -Percent On | Off

An optional switch and literal pair that specify to report individual fault classes as an integer or as a percentage. By default, the values of the individual fault classes are reported as a percentage. When this switch is set to off, fault classes are reported as an absolute number.

- -SET_Baseline

An optional switch that specifies for the tool to store the current coverage information as a baseline so that subsequent report_udfm_statistics commands can add information about the coverage gain. Note that the baseline becomes invalid if the fault list is modified, for

example, by an [add_faults](#) or [delete_faults](#) command. The baseline also becomes invalid if you subsequently specify a different pathname with the -Instance switch.

Examples

Example 1

In this example, the command generates a detailed fault coverage report based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU.

report_udfm_statistics

Note

 Please note that due to space some fault class columns in the report have been removed.

UDFM Fault Statistic														
Type	Block	Name	Instances	Port-Insts	Flts	DS[%]	DI[%]	PT[%]	TI[%]	RE[%]	AU[%]	UC[%]	FC[%]	
TC[%]														
intrac	Cell	OR	518	0	5000	0.00	0.10	0.00	0.00	0.00	0.00	0.20	99.70	0.10
0.10														
intrac	Cell	NOR	139	0	1300	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	XNOR	346	0	12000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	NAND3	387	0	3800	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	NAND4	71	0	600	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	AND2	913	0	10000	0.00	0.00	0.00	0.00	0.00	0.00	1.10	97.60	0.00
0.00														
intrac	Cell	AND3	27	0	300	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	AND4	19	0	400	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
intrac	Cell	AO112	4	0	100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
0.00														
		Total	2424	0	33500	0.00	0.02	0.00	0.00	0.00	0.36	99.19	0.02	
0.02														

Example 2

In this example, the command generates a fault coverage report in which all UDFM fault models that contain the “NAN” string in their name are reported as a single row and all that contain the “AND” string in their name are reported as a single row. All others are reported one row per model.

report_udfm_statistics -groups *NAN* AND*

```
UDFM Fault Statistic
Type   Block Name Instances Port-Insts Faults DS[%] DI[%] PT[%] TI[%] RE[%] AU[%] UC[%]
FC[%] TC[

-----
-- 
intrac Cell OR      518       0 5000 0.00 0.10 0.00 0.00 0.00 0.00 0.20 99.70
0.10 0.10
intrac Cell NOR     139       0 1300 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00
0.00 0.00
intrac Cell XNOR    346       0 12000 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00
0.00 0.00
intrac Cell AO112   4         0 100 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00
0.00 0.00
intrac Cell *NAN*   458       0 4400 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00
0.00 0.00
intrac Cell AND*    957       0 10700 0.00 0.00 0.00 0.00 0.00 0.00 1.09 97.60
0.00 0.00
          Total    2424      0 33500 0.00 0.02 0.00 0.00 0.00 0.00 0.36 99.19
0.02 0.02
```

Example 3

The following example first sets a baseline for later reporting the fault coverage gain. The output of the second command displays two additional columns on the right that list the coverage gain.

```
report_udfm_statistics -set_baseline
...
report_udfm_statistics

UDFM Fault Statistics
// 
Type   Block Name Instances Port-Insts Faults DS[%] ... TC[%] Add-DT GN[%]
-----
-- 
intra_cell Cell AO      1801      0 142918 99.36 99.91 5162 3.63
intra_cell Cell MUXI4   332       0 35856 100.00 100.00 1514 4.22
intra_cell Cell XOR3    50        0 2350 98.89 100.00 21 0.90
intra_cell Cell AND     959       0 11660 97.64 98.96 19 0.17
intra_cell Cell BF      1653      1653 8265 74.41 95.06 11 0.13
...
intra_cell Cell MUXI2   204       0 4360 99.27 99.63 9 0.21
intra_cell Cell IV      1356      1356 6780 88.79 98.02 9 0.13
intra_cell Cell OR      515       0 7745 92.46 98.49 1 0.01
port_defects Cell DF    864       13 5220 1.02 94.61 0 0.00
          Total    33886     3022 724779 87.71 95.69 10763 1.50
```

Related Topics

[write_faults](#)

report_version_data

Context: unspecified, all contexts

Mode: all modes

Displays the current software version information.

Usage

```
report_version_data [{> | >>} file_pathname]
```

Description

Displays the current software version information.

The report_version_data command displays information relating to the software title, version, and date.

Arguments

- `>file_pathname`
An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.
- `>>file_pathname`
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following is an example of the output this command:

```
report_version_data
```

```
Version data: Tesson Shell 2012.3   Wed Sep 05 21:31:33 GMT 2012
```

report_wrapper_cells

Tools Supported: DFTAdvisor

Scope: Dft

Context: dft -scan

Mode: analysis

Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.

Usage

```
report_wrapper_cells [-Summary | -Verbose]
```

Description

Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.

If you specify no arguments, the tool reports the following information for each I/O:

- Primary I/O port name and its direction (Input or Output).
- Total number of sequential cells identified for this I/O.
- Whether the IO's wrapper cell will be a dedicated wrapper cell or an existing (shared) cell.
- The reason the cell was promoted to being a shared wrapper cell, or if wrapper cell identification failed the reason for the failure. For a description of all possible failures see [Table 5](#)Table 1-1

If you specify -Verbose, in addition to the above information, the tool reports the following:

- Identified wrapper cell path names for each I/O that succeeded the identification or “new cell” for the I/Os that failed wrapper cell identification and are subject to I/O registration.
- Wrapper chain type indicates whether the corresponding identified wrapper cell is part of the Input or Output wrapper chain.
- Clocks controlling both the identified wrapper cells and wrapper cells to be added.

If you specify -Summary, the tool reports the following:

- Total number of primary inputs.
- Total number of primary outputs.
- Total number of identified wrapper cells.
- Total number of added wrapper cells.

Table 5-18. Reason for Failed Identification

Reason Failed Identification	Description
Max Logic Level	Exceeded the integer limit on the number of combinational logic levels between this PI/PO and the first level of sequential cells.
Max Fan In	Exceeded the integer limit on the total number of fanins at the instance.
Max Fan Out	Exceeded the integer limit on the total number of fanouts at the instance.
Combinational Feed-thru	Encountered no sequential cells or combinational logic other than buffers/inverters before reaching PO/PI: port_name.
Seq Feed-thru	Encountered no combinational logic and just one sequential cell before reaching PO/PI: port_name.
Combinational Logic Only	Encountered combinational logic only before reaching PO/PI: portname.
Black Box Instance	Encountered a blackboxed instance: instance_name.
Pre-Existing Scan Cell	Encountered a sequential cell belonging to a pre-existing scan chain: instance_name.
Non-Scan Sequential Cell	Encountered a non-scan sequential cell: instance_name
Input Wrapper Cell	Encountered a sequential cell that was already identified as an output or input wrapper cell: instance_name.
Max Identified Cells	Exceeded the integer limit on the number of sequential cells identified per PO/PI.
Max Combinational Gates	Exceeded the integer limit on the number of combinational logic gates traced per PO/PI.
No Cells Identified	Unconnected or floating PO/PI.
Scan-in Pin	Port is a scan-in pin. Port is in the I/O registration list and will be registered.
Scan-out Pin	Port is a scan out pin. Port is in the I/O registration list and will be registered.
Driving a Scan-in	Port is driving a scan-in pin pin_name. Port is in the I/O registration list and will be registered.
Driven by a Scan-out	Port is driven by a scan-out pin pin_name. Port is in the I/O registration list and will be registered.
Input Subchain	Encountered a sequential cell belonging to a subchain that was assigned to an Input wrapper chain: inst-name.

Table 5-18. Reason for Failed Identification (cont.)

Reason Failed Identification	Description
Output Subchain	Encountered a sequential cell belonging to a subchain that was assigned to an Output wrapper chain: inst-name.
Subchain or Multibit Cell	Encountered a subchain or multibit cell with scan cells inside it not reachable from PIs/POs.
Input Subchain or Multibit Cell	Encountered a subchain or multibit cell that was already identified as an output wrapper cell.
Output Subchain or Multibit Cell	Encountered a subchain or multibit cell that was already identified as an input wrapper cell.
Not Traced Subchain Cell	Encountered a sequential cell belonging to a subchain whose internal scan path is not known: inst-name.
Unidentified failure	The reason for failure cannot be identified.

Arguments

- **-Summary**
An optional switch. For information about this switch, refer to the above description.
- **-Verbose**
An optional switch. For information about this switch, refer to the above description.

Examples

Example 1

The following example shows the output generated when the report_wrapper_cells command is executed without the -verbose or -summary switch.

```
report_wrapper_cells
> report_wrapper_cells
-----
Primary I/O # Shared Wrapper Cells      Shared/      Reason
Port          (Direct/Internal-Feedback) Dedicated
              Identified [32/32]           Wrapper Cell
-----
'po' (O)     1                         shared       --
'pi' (I)     1/0                       shared       --
'cen4' (I)   1/0                       shared       clock_gater promoted
'cen5' (I)   1/0                       shared       clock_gater promoted
-----
```

Example 2

The following example shows the output generated when the report_wrapper_cells command is executed with the -summary switch.

```
report_wrapper_cells -summary
```

Command Dictionary (R) **report_wrapper_cells**

```
> report_wrapper_cells -summary
-----
Inputs   Outputs  Shared Wrapper Cells  Inferred Dedicated
(#[#)      (#[#)    Identified (#[#)        Wrapper Cells (#[#)
-----
17       5         3                   0
-----
```

Example 3

The following example shows a snippet of the output generated when the report_wrapper_cells command is executed with the -verbose switch.

report_wrapper_cells -verbose

```
> report_wrapper_cells -verbose
-----
Primary I/O # Shared Wrapper Cells      Wrapper  Wrapper  Clock     Shared/      Reason
Port          (Direct/Internal-Feedback) Cell     Cell      Type
Identified [32/32]           Name      Type
-----
'po' (O)    1           'ff1'    input    'gclk1'  shared    --
'pi' (I)    1/0        'ff1'    input    'gclk1'  shared    --
'cen4' (I)  1/0        'ff4'    output   'clk4'   shared
clock_gater promoted
'cen5' (I)  1/0        'ff5'    output   'clk5'   shared
clock_gater promoted
-----
The wrapper cells identification has failed for the following PIs and POs.
Dedicated wrapper cells will be added to these PIs and POs unless specifically excluded
by the user.
-----
'pi4'           input                           Encountered
combinational logic only before reaching PO: 'pol'
                                         No dedicated
wrapper cells will be added to this PI based on user specifications.
'pol'           output                          Encountered
combinational logic only before reaching PI: 'pi4'
                                         No dedicated
wrapper cells will be added to this PO based on user specifications.
-----
No dedicated wrapper cells will be added to these PIs and POs unless explicitly specified
by the user.
-----
'cen1'           input                           Constrained port
'clk1'           input                           Clock port
'pil'            input                           Unconnected or
blocked PI
'po2'           output                          Unconnected or
blocked PO
-----
```

report_write_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the currently defined write control lines and their off-states.

Usage

`report_write_controls [> | >>] file_pathname]`

Description

Displays the currently defined write control lines and their off-states.

The report_write_controls command displays the write control lines, with corresponding off-states, added using the add_write_controls command.

Arguments

- `> file_pathname`
An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.
- `>> file_pathname`
An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example adds four write control lines and then displays a list of the control line definitions:

```
add_write_controls 0 w1 w3
add_write_controls 1 w2 w4
report_write_controls
```

Related Topics

[add_write_controls](#)
[analyze_control_signals](#)
[delete_write_controls](#)

report_write_patterns_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Displays the current set of ports or current set of vector callbacks to use when writing patterns.

Usage

Usage for Port Lists

```
report_write_patterns_options [-existing_used_ports | -additional_port_list  
| -all_used_port_list]
```

Usage for Vector Callbacks

```
report_write_patterns_options [-vector_callbacks]
```

Description

This command displays the current set of ports or current set of vector callbacks to use when writing patterns.

Port List Specifics

This command and options are used to display the current set of ports to use when writing patterns. The `-existing_used_ports` option displays only the current subset of existing design ports. The option `-additional_ports` displays only the additional ports that were not part of the original design ports. The `all_used_ports` option reports the current subset of existing design ports and any extra ports added with `-additional_ports`. The report is a tabular format which lists the port names, port direction the port is an additional port, and if the port is a clock and what its offstate is if it is a clock. If no subset exists, a message stating that there is no subset is issued, but the full list of default ports is then listed. If the `report_write_pattern_options` command is issued without any options, all `write_pattern_options` settings are reported (the `all_used_ports` and the vector callbacks).

Vector Callback Specifics

This command reports the current vector callback procs listing each callback proc name followed by a textual representation of any activation trigger. If the `report_write_pattern_options` command is issued without any options, all `write_pattern_options` settings are reported (the used ports and the vector callbacks).

Arguments

- `-existing_used_ports`
An optional switch that reports only the current subset of the existing design ports.
- `-additional_port_list`
An optional switch that reports only the additional ports that were not part of the original design ports.

- **-all_used_port_list**
An optional switch that reports the current subset of existing design ports and any extra ports added with -additional_ports.
- **-vector_callbacks**
An optional switch that reports the current vector callback procs.

Related Topics

[get_write_patterns_options](#)

[set_write_patterns_options](#)

[write_patterns](#)

[Vector Creation and Modification](#)

report_xbounding

Context: dft -scan, dft -test_points

Mode: analysis

Reports the X-sources and optionally the scan cells used in bounding. You must issue this command to report X-bounding.

Usage

```
report_xbounding {[ -verbose { all | non_mcp_bounding_points | mcp_bounding_points } ]}  
[ -ignored_x_sources { on | off } ] [> | >> file_pathname]
```

Description

Reports the X-sources and optionally the scan cells used in bounding. You must issue this command to report X-bounding.

By default, this command reports a summary that includes the number of identified X-sources and also the number of control points that are required to block these X-sources from reaching any scan cells (or potential scan cells). The command can also be used to report the location of the X-bounding logic (typically muxes) as well as the control signals for the bounding logic.

Arguments

- **-verbose all | non_mcp_bounding_points | mcp_bounding_points**
An optional switch and optional literal that reports the location of the scan cells used in the bounding.
Choose one from the following:
 - all** — Reports all scan cells used in the bounding including non-MCP and MCP bounding points. This is the default.
 - non_mcp_bounding_points** — Reports only non-MCP bounding points.
 - mcp_bounding_points** — Reports only MCP bounding points.
- **-ignored_x_sources { on | off }**
An optional switch and literal that specifies to report X-sources that do not need to be bounded, and the reasons why they will not be bounded. The default is off.
- **> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.
- **>> *file_pathname***
An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the tool's verbose output:

```
report_xbounding -verbose
// Analyzed 52 static x-sources and 1 flop that captures false/
// multi-cycle paths.
// Added 53 control points for x-bounding.
//      52 control points bounding static x-sources.
//      1 control point bounding false or multi-cycle paths.
MUX gate at /NEA(3) bounding mux is driven by existing flop at : /u12/
L_MSIZ_reg_7_/Q (326)
MUX gate at /NESFR(4) bounding mux is driven by existing flop at : /u12/
u1/L_SP_reg_0_/Q (323)
```

Related Topics

[analyze_xbounding](#)
[insert_test_logic](#)
[set_xbounding_options](#)
[read_sdc](#)

reset_attribute_value

Context: unspecified, all contexts

Mode: all modes

Resets an attribute to its default value for the design objects specified in *obj_spec*.

Usage

```
reset_attribute_value { obj_spec | -object_types type_list} -name attribute_name [-silent]
```

Description

Resets an attribute to its default value for the specified design objects or design object types.

Resetting an attribute differs from setting it to its default value in the fact that the *reset_attribute_value* also makes the attribute unspecified so it does not show up when the *report_attributes* and *get_attribute_list* commands are executed using Usage 1. The attribute's default value is specified during registration of the attribute.

This command returns the default value set for the attribute as a string.

If the *attribute_name* does not exist, or if a design object or object type does not exist, an error message displays unless *-silent* is specified.

Arguments

- ***obj_spec***

A switch that specifies a Tcl list of one or more object names or a collection of one or more objects.

- **-object_types *type_list***

A switch that specifies a list of object types for which the given attribute name is to be reset. As an example, using “*reset_attribute_value -name XXX -object_type instance*” is equivalent to specifying “*reset_attribute_value -name XXX [get_instances]*” but is more efficient and requires less memory and time when the number of objects for the given object type is large.

- **-name *attribute_name***

A required switch and value pair that specify the name of the attribute.

- **-silent**

An optional switch that specifies to suppress error messages if the *attribute_name* does not exist, or if an object in *obj_spec* does not exist.

Examples

Example 1

The following example shows how the command resets the added_by values for all nets to its default (“John”):

```
reset_attribute_value {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by  
get_attribute_value_list {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by  
{John John John John}
```

Example 2

The following example resets the my_constraint attribute on all pins and nets.

```
reset_attribute_value -name my_constraint –object_types {pin net}
```

Related Topics

[get_attribute_list](#)
[get_attribute_option](#)
[get_attribute_value_list](#)
[report_attributes](#)
[set_attribute_options](#)
[set_attribute_value](#)
[unregister_attribute](#)

reset_au_faults

Context: dft -edt, patterns -scan

Mode: analysis

Reclassifies the faults in certain untestable categories.

Usage

```
reset_au_faults [-REClassify]
```

Description

Reclassifies the faults in certain untestable categories.

Table 5-19 shows these categories.

Table 5-19. Untestable Faults that are Reclassified by reset_au_faults

Untestable Fault	Fault Reclassification
Possibly detected untestable (PU)	Possibly detected testable (PT)
ATPG untestable (AU) except for AU.LBIST (not reclassified)	Uncontrolled (UC)

Deterministic fault simulators classify some untestable faults differently depending on the algorithm. You can use the reset_au_faults command to align those (potentially) misclassified faults.

When the command changes the fault classification, the tool then has the ability to analyze and further reclassify each previously-untestable fault. Allowing the tool the ability to analyze and reclassify those particular untestable faults increases the tool's efficiency.

Arguments

- **-REClassify**

An optional switch that analyzes the AU faults to identify those which cannot be tested with the current settings. After resetting all AU faults, the tool will reclassify faults that are still untestable in the current mode back to AU. This switch is useful when reading in faults from a previous ATPG run, which has different settings or a different design configuration.

Reclassifying the faults to AU prior to pattern generation will improve performance. If you omit this switch, the tool does not reclassify the faults and resets all AU faults. See Example 2.

Examples

Example 1

The following example sets up the tool to run the simulation with an external pattern file and resets the ATPG untestable faults so that the tool can determine their appropriate fault category:

```
read_patterns testpatterns
read_faults /user/design/fault_file -retain
reset_au_faults
simulate_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage.

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault -retain
reset_au_faults -reclassify
create_patterns
read_faults mode1.fault -merge
report_statistics
write_faults modes_1_and_2.fault -replace
```

Related Topics

[read_faults](#)
[reset_di_faults](#)

reset_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes user-specified EDT bypass chain connections.

Usage

```
reset_bypass_chains
```

Description

Removes user-specified EDT bypass chain connections.

The `reset_bypass_chains` command removes all EDT bypass chain connections that you previously defined using the `set_bypass_chains` command, and reverts determination of these connections to the tool.

Arguments

None

Examples

The following example displays information on all user-defined EDT bypass chains in a design with eight scan chains and two scan channels. The example then removes the definitions, and confirms that configuration of any EDT bypass chains will now be done automatically by the tool.

```
report_bypass_chains
//      Reporting user-defined bypass chains:
//
//      Bypass#  EDT chains
//      -----  -----
//      1        chain1, chain5, chain6
//      2        chain2, chain3, chain4, chain7, chain8

reset_bypass_chains
report_bypass_chains
//  No user-defined bypass chains to report.
```

Related Topics

[report_bypass_chains](#)

[set_bypass_chains](#)

reset_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes user-specified EDT compactor connections.

Usage

```
reset_compactor_connections
```

Description

Removes user-specified EDT compactor connections.

The `reset_compactor_connections` command removes all EDT compactor connections that you previously defined using the `set_compactor_connections` command, and reverts determination of these connections to the tool.

Arguments

None

Examples

The following example displays information on all user-defined compactor connections in a design with eight scan chains and two scan channels. The example then removes the user-specified definitions. Configuration of compactor connections will now be handled automatically by the tool.

```
report_compactor_connections
//      Reporting user-defined compactor connections:
//      Channel  Chains
//      -----  -----
//      1        chain1, chain2, chain6, chain7, chain8
//      2        chain3 - chain5
```

```
reset_compactor_connections
report_compactor_connections
//  No user-defined compactor connections to report.
```

Related Topics

[report_compactor_connections](#)
[set_compactor_connections](#)

reset_design

Context: all contexts

Mode: setup

Clears all design setup data but keeps the design and library in memory.

Usage

`reset_design`

Description

Clears all design setup data but keeps the design and library in memory.

The command also retains the setting from the most recent [set_current_design](#) command execution.

Arguments

None

Related Topics

[delete_design](#)

[set_current_design](#)

reset_di_faults

Context: dft -edt, patterns -scan

Mode: analysis

Reclassifies DI faults to the UC category.

Usage

```
reset_di_faults {-ALL | -END_of_chains | -CHains [chain_name...]} | pin_pathname... }  
[-Stuck_at {01 | 0 | 1}] [-Both | -Rise | -Fall] [ -CELL_Output_only ]
```

Description

Reclassifies DI faults to the UC category.

The reset_di_faults command reclassifies det_implication (DI) faults to the uncontrolled (UC) category. When the fault type is stuck-at or transition, this command by default resets both stuck-at-0 (or slow-to-rise for transition faults) and stuck-at-1 (or slow-to-fall for transition faults) faults at each DI fault site. The command does not reclassify DI faults identified by other Tessonnt plugins, such as DI.MBIST and DI.MBISR faults identified by Tessonnt MemoryBIST. You can use one of the optional switches to reset just one, rather than both, faults at each DI fault site.

Use this command when you want faults that were previously classified as DI (and thus no longer targeted for detection) retargeted in the next ATPG and/or fault simulation run. For example, the tools currently declare scan path faults DI because they are tested by the chain test. This is not optimal in some cases; the end of each scan chain (after the last scan cell) is often used as a system path and it is sometimes desirable to create an at-speed transition fault test to exercise this path. The DI fault category prevents ATPG from targeting or simulating such faults. When the fault classification is changed to UC, the tool then has the ability to target the faults during ATPG and fault simulation.

Arguments

- **-ALL**
A switch that specifies for the tool to reset DI faults to UC. This is the invocation default.
- **-END_of_chains**
A switch that specifies to reset, for each scan chain, the scan path DI faults from the output of the last scan cell in the chain to the scan output pin.
- **-CHains [chain_name...]**
A switch and optional repeating string pair that resets DI faults, including scan input and output pin faults. If you do not specify a chain_name, the tool resets faults along all scan chains. You can optionally restrict the switch operation to one or more chain_names.

- ***pin.pathname...***

A repeating string that resets the DI faults on one or more pins. The *pin.pathname* must be an instance/pin name that exists in the flat model. These are typically model boundary pins, PIs, and POs.

- **-Stuck_at 01 | 0 | 1**

An optional switch and literal pair that specifies which DI faults to reset to UC. You should use this switch only if the fault type is stuck, although the tool accepts the switch regardless of fault type (see [set_fault_type](#)).

01 — A literal specifying that for stuck-at faults the tool reset both the “stuck-at-0” and “stuck-at-1” faults; or for transition faults the tool reset both “slow-to-rise” and “slow-to-fall” faults. This is the default.

0 — A literal specifying that for stuck-at faults the tool reset only the “stuck-at-0” faults; or for transition faults the tool reset only “slow-to-rise” faults.

1 — A literal specifying that for stuck-at faults the tool reset only the “stuck-at-1” faults; or for transition faults the tool reset only “slow-to-fall” faults.

- **-Both | -Rise | -Fall**

Optional switches that specify which transition faults to reset to UC. These switches apply to transition faults only:

-Both - An optional switch that specifies to reset both slow-to-rise and slow-to-fall transition faults. This is the default.

-Rise - An optional switch that specifies to reset only the slow-to-rise transition faults.

-Fall - An optional switch that specifies to reset only the slow-to-fall transition faults.

- **-CELL_Output_only**

Optional switch that reclassifies DI faults only on the scan output ports of scan cells.

Examples

Example 1

The following example resets all stuck-at-0 and stuck-at-1 DI faults (including clock faults) to UC.

reset_di_faults

Example 2

The following example resets all stuck-at-0 faults declared DI anywhere in the netlist.

reset_di_faults -stuck_at 0

Example 3

The following example resets stuck-at-0 and stuck-at-1 DI faults at the output of the last cell of each chain.

reset_di_faults -end_of_chain

Example 4

The following example resets only the slow-to-rise transition fault at the output of the last cell of each chain.

`reset_di_faults -end_of_chain -rise`

Example 5

The following example resets any DI fault anywhere along chain1.

`reset_di_faults -chain chain1`

Example 6

The following example resets both DI faults at pin /top/mid/bottom/Q (any type fault).

`reset_di_faults /top/mid/bottom/Q -stuck_at 01`

Example 7

The following example is equivalent to the preceding example, except that it applies to transition faults only.

`reset_di_faults /top/mid/bottom/Q -both`

Example 8

The following example uses the defaults to reset_di_faults at the specified location (any fault type).

`reset_di_faults /top/mid/bottom/Q`

Example 9

The following example resets only the slow-to-rise transition fault at the specified location.

`reset_di_faults /top/mid/bottom/Q -rise`

Example 10

The following example resets only the stuck-at-1 fault (or slow-to-rise if transition fault) at the specified location.

`reset_di_faults /top/mid/bottom/Q -stuck_at 1`

Related Topics

[read_faults](#)

[reset_au_faults](#)

reset_open_pattern_set

Context: patterns -ijtag

Mode: analysis

Clears the content of the currently open pattern set.

Usage

```
reset_open_pattern_set
```

Description

Clears the content of the currently open pattern set.

After this command executes, the pattern set remains opened. The state of the ICL network is restored to the state it was in at the beginning of the pattern set which is either the end state of the previously closed pattern set when the [open_pattern_set](#) -no_initial_ireset option is used or the reset state.

Arguments

None

Return Values

None

Examples

The following example clears the content of the currently open pattern set.

```
reset_open_pattern_set
```

Related Topics

[close_pattern_set](#)[get_open_pattern_set](#)[get_pattern_set_list](#)[open_pattern_set](#)[report_pattern_sets](#)

reset_state

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Resets the circuit status.

Usage

```
reset_state
```

Description

Resets the circuit status.

Resets all faults, except AU, TI, BL, and RE faults, to be undetected in order to run a new simulation; and deletes the internal patterns.

dft -scan only:

Removes all instances from both the scan identification and test point identification lists that the tool identified.

Arguments

None

Examples

The following example first performs an ATPG run, then resets the circuit status, resets the faults to be undetected, deletes the internal patterns, and then performs a new ATPG run:

```
set_system_mode analysis
create_patterns
reset_state
create_patterns
```

reset_static_dft_signal_values

Context: dft patterns

Mode: setup, analysis

Resets the static DFT signal values that were previously set using the `set_static_dft_signal_values` command.

Usage

```
reset_static_dft_signal_values
  [-all | -current_design | -icl_instances icl_instances | -instances instances]
  [{dft_signal_name ...}]
```

Description

Resets the static DFT signals that were previously set using the [set_static_dft_signal_values](#) command. All static DFT signals implemented with IJTAG can be set using this command. See the [add_dft_signals](#) command for information on how to add the DFT signals. The DFT signals are grouped by each module in which you added DFT signals and ran the `process_dft_specification` command. These modules have the `tessent_design_level` attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value following the logic documented in the [set_static_dft_signal_values](#) command description. The `reset_static_dft_signal_values` command is used to remove an explicit setting. If applicable, the implied settings are also removed.

Arguments

- **-all**

An optional switch that removes the settings on the specified DFT signals found within all ICL instances that have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also reset.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-current_design**

An optional switch that removes the settings on the specified DFT signals found within top module. The top-level module must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within its ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-icl_instances *icl_instances***

An optional switch that removes the settings on the specified DFT signals found within the specified ICL instances. The specified ICL instances must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-instances *instances***

An optional switch that removes the settings on the specified DFT signals found within the specified design instances. The specified design instances must match the value of the “`hierarchical_design_instance`” attribute found on an ICL instance with the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- ***dft_signal_name***

A repeatable set of strings used to specify DFT signal names to reset. The string can also be a single well-formatted Tcl list containing DFT signal names. If you specify no `dft_signal_name`, then the tool resets all DFT signals.

Examples

The following example shows the use of the `report_static_dft_signal_settings` command after a signal was set using the `set_static_dft_signal_values` and after the same setting was reset using the `reset_static_dft_signal_values` command.

```
set_context patterns -ijtag
read_icl ./sub_block_l3.icl
set_current_design sub_block_l3

set_system_mode analysis
set_static_dft_signal_values int_edt 1
report_static_dft_signal_settings

// ICL Module      : sub_block_l3

// -----  -----
// DFT Signal Name Usage          Set value  Set source
// -----  -----
// all_test        global_dft_control    1       Inferred
// ltest_en        logic_test_control   1       Inferred
// int_ltest_en   logic_test_control   1       Inferred
// ext_ltest_en   logic_test_control   -       -
// int_edt         scan_mode(internal)  1       Explicit
// ext_multi       scan_mode(external) -       -
//
```

```
reset_static_dft_signal_values int_edt
report_static_dft_signal_settings

// ICL Module      : sub_block_l3

// ----- -----
// DFT Signal Name Usage          Set value Set source
// ----- -----
// all_test          global_dft_control   -       -
// ltest_en          logic_test_control    -       -
// int_ltest_en     logic_test_control    -       -
// ext_ltest_en     logic_test_control    -       -
// int_edt          scan_mode(internal)   -       -
// ext_multi         scan_mode(external)  -       -
```

Related Topics

[get_static_dft_signal_icall](#)
[report_static_dft_signal_settings](#)
[set_static_dft_signal_values](#)
[add_dft_signals](#)

restore_design

Context: dft

Mode: insertion

Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.

Usage

`restore_design`

Description

Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.

Arguments

None

Examples

The following example creates a minor edit to the design, and then discards the edit:

```
SETUP> set_context dft -rtl
SETUP> read_verilog design1
SETUP> get_ports
{i1 i2}

SETUP> set_system_mode insertion
INSERTION> create_port i3
{i3}

INSERTION> get_ports
{i1 i2 i3}

INSERTION> restore_design
INSERTION> get_ports
{i1 i2}
```

Related Topics

[write_design](#)

run_synthesis

Mode: all contexts

Context: all modes

Invokes a synthesis manager to perform synthesis of the test logic RTL.

Usage

```
run_synthesis [-design_name design_name] [-design_id design_id] [-generate_scripts_only]  
[-synthesis_tool dc_shell] [-compilation_options option_list]  
[-pre_compilation_commands command_list]  
[-post_compilation_commands command_list]  
[-synthesis_output_directory directory] [-startup_file file_name]  
[-wait]
```

Description

Invokes a synthesis manager to perform synthesis on test logic RTL. The [check_synthesis](#) command monitors the synthesis results. Additionally, the [set_run_synthesis_options](#) command specifies the default values for options that can be set for run_synthesis command. The options that are defined with set_run_synthesis_options are overridden if they are explicitly defined with the run_synthesis command.

The run_synthesis command only synthesizes test logic RTL contained within the TSDB. When creating and inserting memory BIST, boundary scan or IJAG logic, the generated RTL is automatically written to the TSDB during process_dft_specification. With EDT logic RTL, the [write_edt_files](#) with the -tsdb option is required to create a TSDB that can be used with run_synthesis.

When the run_synthesis is used with a gate level design, a concatenated netlist of the design which contains the synthesized test logic and modified design modules will automatically be created and placed in the dft_inserted_designs directory of the TSDB.

This command is typically run with no arguments. It automatically uses the design name and design id from the previously processed DftSpecification(*design_name*, *design_id*) wrapper.

The command dispatches and connects to a synthesis manager using a socket. If a manager is already running, it simply connects to it. The synthesis manager is an unlicensed Tcl shell executable that uses the content found in the *design_name_design_id.synthesis_dc_shell* directory to synthesize the test logic for the given *design_name* and *design_id*.

[Figure 5-15](#) shows the subdirectories and files created with run_synthesis. Descriptions of their content and usage are also described. One synthesis manager is created per user/host combination. It remains alive even if Tesson Shell is terminated or suspended. The manager terminates itself when it has finished dispatching all of the simulations and Tesson Shell is not attached to it through a socket.

Figure 5-15. Typical Contents of the synthesis_outdir Directory

```
synthesis_outdir
|-- design_name_design_id.synthesis_dc_shell
|-- design_name_design_id.cells.instrument
|   |-- design_name_design_id_tessent_*.vg
|-- design_name_design_id_ijtag.instrument
|   |-- design_name_design_id_tessent_*.vg
|-- design_name_design_id_mbisr.instrument
|   |-- design_name_design_id_tessent_mbisr_register_*.vg
|-- design_name_design_id_mbist.instrument
|   |-- design_name_design_id_tessent_mbist_bap.vg
|   |-- design_name_design_id_tessent_mbist_*_controller.vg
|   |-- design_name_design_id_tessent_mbist_*_retiming_*.vg
|-- command.log
|-- synthesis_tool.synthesis_log
|-- synthesis_tool.synthesis_tcl
|-- file_copy.script
|-- file_list
|-- synthesis.script
```

- *design_name_design_id.synthesis_dc_shell* — A directory that holds the synthesis data associated with a given *design_name* and *design_id*.
- *design_name_design_id.cells.instrument* — A directory that holds the synthesized rtl cells associated with a given *design_name* and *design_id*.
- *design_name_design_id_ijtag.instrument* — A directory that holds the synthesized IJTAG RTL instruments associated with a given *design_name* and *design_id*.
- *design_name_design_id_mbisr.instrument* — A directory that holds the synthesized BISR RTL instruments associated with a given *design_name* and *design_id*.
- *design_name_design_id_mbist.instrument* — A directory that holds the synthesized memory BIST RTL instruments associated with a given *design_name* and *design_id*.
- *command.log* — A file that contains a transcript of the commands executed by the synthesis tool.
- *synthesis_tool.synthesis_log* — A file that contains the complete transcript that is generated when running the *synthesis.script*. The synthesis manager simply redirects the output of the script into this file. The [check_synthesis](#) command parses this file and reports the status.
- *synthesis_tool.synthesis_tcl* — A file that contains the Tcl script that this used to perform the synthesis by the *synthesis_tool* invoked by the *synthesis.script*.
- *file_copy.script* — A file that contains the script to manually copy the synthesized modules back to the TSDB.
- *file_list* — A file that contains the list of files used by the *file_copy.script*.
- *synthesis.script* — A file that contains the commands to invoke *synthesis_tool* with the *synthesis_tool.synthesis_tcl* that synthesizes the test logic RTL.

Environment Variables

Synthesis Script Variables

For the run_synthesis command, a list of environment variables is exported to a side shell script that is used by the synthesis scripts when they are executed. This way, the execution should be consistent even when executed standalone, without the simulation/synthesis managers.

This is particularly important because the environment variables can be changed in Tesson Shell. When changed this way, the modified environment variables were automatically inherited when sub-invoking a shell command with the 'exec' or 'system' command. However, since those environment variable were not exported to the simulation/synthesis scripts, the behavior could be different than when sub-invoking the scripts directly with 'exec' or 'system' from Tesson Shell if some key environment variables were modified inside Tesson Shell.

By default, the following list of environment variables are exported:

- VCS_ARCH_OVERRIDE
- PATH
- LD_LIBRARY_PATH
- *_HOME
- *_VERSION
- *_LICENSE_FILE

In addition to the variables in that list, more environment variables can be exported by specifying a list of names ('*' accepted) in an environment variable.

For synthesis, environment variable 'TESSENT_SYNTH_ENVVAR_LIST' is used.

For example, in Tesson Shell, using:

```
setenv TESSENT_SYNTH_ENVVAR_LIST [list MY_VARS_* ANOTHER_VAR]
```

will pick-up and export any environment variable with names that start with 'MY_VARS_' and the variable name 'ANOTHER_VAR'. When a variable name in those lists is optionally prefixed by a '-' sign, it won't be exported. The optional '+' sign is accepted and also, it is the default behavior.

A side file is generated that includes the export of all the selected environment variables and that file is sources by the main synthesis scripts. The file is named *setup.script* and exported to the following location:

synthesis_outdir/design_name_design_id.synthesis_tool

VCS_ARCH_OVERRIDE Variable

The VCS_ARCH_OVERRIDE environment variable is automatically exported in the generated *synthesis.script* with the 'linux' value only when the variable is not set in the setup script and the operating system is detected as 'linux'.

In order to change this default behavior, issue the following command prior to the calling the *run_synthesis* command:

```
setenv VCS_ARCH_OVERRIDE ""
```

Synthesis Logfile Variable

When the environment variable TESSENT_SYNTH_MANAGER_ENABLE_LOG is set to the value 1, a log file is produced that contains the information exchanged between Tesson Shell and the synthesis manager. By default, the name of the file is './tesson_synth_manager.log' and the path and name of this file can be specified by the environment variable TESSENT_SYNTH_MANAGER_LOG_FILE_NAME.

By default, the file is overwritten for every Tesson Shell session. It is possible to append to an existing file by changing the file name and specifying the letter 'a' after the file name. The optional letter 'w' reproduces the default behavior.

For example, the following enables the log append to the specified file, if it already exists:

```
setenv TESSENT_SYNTH_MANAGER_ENABLE_LOG 1  
setenv TESSENT_SYNTH_MANAGER_LOG_FILE_NAME './my_tesson_synth_manager.log a'
```

Synthesis Instance Manager

The TESSENT_SYNTH_MANAGER_INSTANCES environment variable can be used to control the number of managers. By default, when this environment variable is not set, the tool creates one manager per user per host. This is the same as setting the value to 1. When you use a value greater than 1, one manager per user per host per Tesson Shell session is started with the manager name being the following:

```
tesson_synth_manager_pid
```

Using this name, it is possible to find the manager in the PID list reported by the OS using the command:

```
ps -F -C tclsh8.5.exe | cat
```

Synthesis Socket Manager

The TESSENT_SYNTH_MANAGER_SOCKET_SELECTION environment variable is used to manage sockets. The default value is "server" which causes the manager to use any socket channel returned by operating system.

The value "auto" means that the tool automatically selects a socket as returned by the operating system and requests that the started manager use it.

Specifying an integer value forces the started manager to use the specified socket, if available.

Synthesis Timeout Manager

The TESSENT_SYNTH_MANAGER_TIMEOUT_SECS environment variable controls how long the tool waits for the socket manager. By default, the run_synthesis command waits for the manager to start and open a socket for 300 sec (5 minutes). You can use this environment variable to change this value.

Synthesis Debug

As an alternative to the enable log environment variable, you can use the TESSENT_DEBUG_CONTROL environment variable to generate a logfile. You set the TESSENT_DEBUG_CONTROL environment variable to the following value:

```
+tessent_synth_manager/log_file
```

or use the following in Tesson Shell:

```
set_debug_control +tessent_synth_manager/log_file
```

It is possible to generate debug information while the managers are starting/running with the following debug controls:

```
+tessent_synth_manager/show_debug_info
```

This information can be of value to Mentor Graphics personnel in the case where a manager refuses to start or does not perform as expected.

Arguments

- **-design_name *design_name***

An optional switch and value pair that specifies the design name to which run_synthesis will be associated with. By default, the name of the current design is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- **-design_id *design_id***

An optional switch and value pair that specifies the design id to which run_synthesis will be associated with. By default, the name of the current design id is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- **-generate_scripts_only**

An optional switch that fully populates the synthesis output directory but does not run the synthesis tool. You can use this option to create the synthesis scripts, and then invoke synthesis using the generated scripts stand alone rather using the synthesis manager.

- **-synthesis_tool dc_shell**

An optional switch and value pair that specifies the synthesis tool you wish to use for synthesis. The default and currently only permitted option is “dc_shell”. Other synthesis tools may be supported in future releases.

- **-compilation_options *option_list***

An optional switch and option list that specifies additional options to be added to the synthesis compile command used in the <synthesis_tool>.synthesis_tcl. An example would be the -boundary_optimization option for dc_shell.

- **-pre_compilation_commands *command_list***

An optional switch and tcl command list that adds the provided commands to <synthesis_tool>.synthesis_tcl so that they are executed before the compilation command is executed by the synthesis tool. Each element of the list becomes a command line in the <synthesis_tool>.synthesis_tcl script.

- **-post_compilation_commands *command_list***

An optional switch and tcl command list that adds the provided commands to <synthesis_tool>.synthesis_tcl so that they are executed after the compilation command is executed by the synthesis tool. Each element of the list becomes a command line in the <synthesis_tool>.synthesis_tcl script.

- **-synthesis_output_directory *directory***

An optional switch and value pair that specifies the synthesis output directory. The default directory is synthesis_outdir and is located in the Tesson Shell invocation directory.

- **-startup_file *file_name***

An optional switch and file path that allows the user to explicitly specify the startup file that will be used by the synthesis tool. The file specified is copied into the actual synthesis directory.

Note

 If a startup file is not specified and a .synopsys_dc.setup file exists in the Tesson run directory, it will automatically be copied into the synthesis run directory. If relative paths are utilized within this setup file, it should be noted the synthesis run directory is two levels below where Tesson Shell is launched and these paths should account for this execution point.

- **-wait**

An optional switch that waits for synthesis task to be completed before returning with a completion message. When running in interactive mode, the command reports a status line

that is updated every second showing at what stage synthesis is current operating, which can be unscheduled, queued, running, pass, or fail, as shown below:

```
...
// Synthesis script creation for 'blockB_rtl' completed.
//
// Submitting
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'
to synthesis manager
// Waiting for the synthesis to complete
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were
copied to the current tsdb output directory.
```

Examples

The following are examples of run_synthesis using various options.

Example 1

This example runs synthesis without any command options, after completing process_dft_specification:

SETUP> run_synthesis

```

// Synthesis script creation for 'blockB_rtl'
// Begin processing of instrument synthesis dictionaries
// Processing containers for instrument type 'rtl_cells'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_cells.instrument/
blockB_rtl_cells.synthesis_dictionary
// Processing containers for instrument type 'ijtag'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_ijtag.instrument/
blockB_rtl_ijtag.synthesis_dictionary
// Processing containers for instrument type 'mbisr'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbisr.instrument/
blockB_rtl_mbisr.synthesis_dictionary
// Processing containers for instrument type 'mbist'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbist.instrument/
blockB_rtl_mbist.synthesis_dictionary
// Done processing of instrument synthesis dictionaries
//
// Writing TCL synthesis script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/dc_shell.synthesis_tcl'
// Writing the synthesis shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'
// Writing the file copy shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_copy.script'
// Writing the synthesized file list
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list'
// Synthesis script creation for 'blockB_rtl' completed.
//
// Submitting
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script' to
synthesis manager

```

Example 2

This example runs synthesis with `-generate_scripts_only` options so that you may run synthesis outside Tesson Shell:

SETUP> `run_synthesis -generate_scripts_only`

```
// Synthesis script creation for 'blockB_rtl'  
// Begin processing of instrument synthesis dictionaries  
// Processing containers for instrument type 'rtl_cells'  
// Synthesis dictionary  
.tsdb_outdir/instruments/blockB_rtl_cells.instrument/  
blockB_rtl_cells.synthesis_dictionary  
// Processing containers for instrument type 'ijtag'  
// Synthesis dictionary  
.tsdb_outdir/instruments/blockB_rtl_ijtag.instrument/  
blockB_rtl_ijtag.synthesis_dictionary  
// Processing containers for instrument type 'mbisr'  
// Synthesis dictionary  
.tsdb_outdir/instruments/blockB_rtl_mbisr.instrument/  
blockB_rtl_mbisr.synthesis_dictionary  
// Processing containers for instrument type 'mbist'  
// Synthesis dictionary  
.tsdb_outdir/instruments/blockB_rtl_mbist.instrument/  
blockB_rtl_mbist.synthesis_dictionary  
// Done processing of instrument synthesis dictionaries  
//  
// Writing TCL synthesis script  
'.synthesis_outdir/blockB_rtl.synthesis_dc_shell/dc_shell.synthesis_tcl'  
// Writing the synthesis shell script  
'.synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'  
// Writing the file copy shell script  
'.synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_copy.script'  

```

Related Topics

[check_synthesis](#)
[get_run_synthesis_options](#)
[report_run_synthesis_options](#)
[set_run_synthesis_options](#)
[write_design_import_script](#)

run_testbench_simulations

Context: all contexts

Mode: all modes

Invokes a simulation manager to run a set of simulation test benches.

Usage

```
run_testbench_simulations
  [-design_name design_name]
  [-design_id design_id]
  [-pattern_id pattern_id]
  [-report_list]
  [-select pattern_name_glob_list]
  [-exclude pattern_name_glob_list]
  [-generate_scripts_only]
  [-parallel_simulations {1 | 2...MAX_INT | maxcpu}]
  [-expected_miscompare_count int]
  [-simulation_id string]
  [-simulator questa | vcs | incisive]
  [-simulation_timeout time | unlimited]
  [-keep_simulation_data on | off | on_failure]
  [-store_simulation_waveforms on | off]
  [-simulation_run_commands simulator_commands]
  [-waveform_configuration_commands simulator_command_list]
  [-compilation_macro_definitions {macro[=value] ... }]
  [-simulation_macro_definitions {macro[=value] ... }]
  [-compilation_options option_list]
  [-simulator_options option_list]
  [-extra_verilog_files list_of_files]
  [-extra_top_modules module_name_list]
  [-simulation_output_directory simulation_output_directory]
  [-simdut_output_directory simdut_output_directory]
  [-simdut_server] [-generate_simdut_files_only]
  [-no_wait]
```

Description

Invokes a simulation manager to run a set of simulation test benches. The [check_testbench_simulations](#) command monitors the simulation results.

This command is typically run with no arguments at all. It automatically uses the design name, design id, and pattern id found in the previously processed [PatternsSpecification\(design_name, design_id, pattern_id\)](#) wrapper.

The information on how to load the design view into the simulator is contained in the *<design_name>.design_source_dictionary* files found inside the [dft_inserted_designs](#)

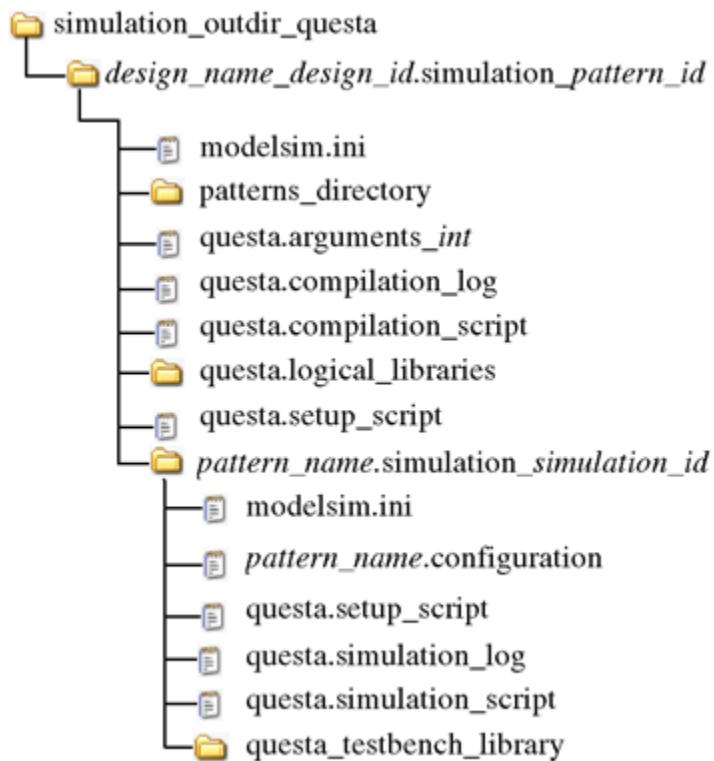
directories. The only missing information is where to find the simulation library models because those are not loaded inside Tesson Shell. You specify the location of the simulation library models using the [set_simulation_library_sources](#) command.

Refer to the “[check_testbench_simulations -report_status](#)” switch for details on obtaining status for a given simulation run.

The command dispatches and connects to a simulation manager using a socket. If a manager is already running, it simply connects to it. The simulation manager is an unlicensed Tcl shell executable that uses the content found in the simulation.data_dictionary file inside the *design_name_design_id.patterns_pattern_id* directory to compile the design and the test benches into the specified -simulation_output_directory, and to simulate them using the specified simulator. The simulations are run in series or in parallel based on the specified -parallel_simulations value. [Figure 5-16](#) shows the subdirectories and files created below the specified -simulation_output_directory. Descriptions of their content and usage are described next.

One simulation manager is created per user/host combination. It remains alive even if Tesson Shell is terminated. The manager terminates itself when it has finished dispatching all of the simulations and Tesson Shell is not attached to it through a socket.

Figure 5-16. Example Output Directory Structure (Questa)



- *design_name_design_id.simulation_pattern_id*

A directory that holds the simulation data associated with a given *design_name*, *design_id*, and *pattern_id* combination.

- *patterns_directory*

A soft link that points back to the directory in which the Verilog test bench files exists. The *simulator.simulation_script* file uses that link to access the test bench file.

- *simulator.arguments_int*

A file created for all each unique set of “-y ... -v ... +libext+...” in the compilation/simulation shell scripts generated. This file is added to the Verilog compilation command as “-f *simulator.arguments_int*”.

- *simulator.compilation_script*

A file that contains simulator-specific commands to map logical libraries to physical directories and compile the complete design into the various logical libraries. An extra logical library called “tsdb_interface” is created and the *design_name.format_interface* files of the first level of physical block modules are compiled into it. The *pattern_name.configuration_files* use this library to bind the interface view for different instances of physical blocks based on what was specified in the LowerPhysicalBlockInstances wrapper inside the [SimulationOptions](#) wrapper.

- *simulator.logical_libraries*

A directory that contains one subdirectory per logical library, and in which the compilation data is stored.

- *simulator.compilation_log*

A file that contains the complete transcript generated when running the *simulator.compilation_script*. The simulation manager simply redirects the output of the script into this file.

- *pattern_name.simulation[_simulation_id]*

A directory that is created to simulate each pattern. The *_simulation_id* extension suffix is only present when the “run_testbench_simulations [-simulation_id string](#)” option is used.

- *simulator_testbench_library*

A directory used to store the compilation data for the test bench and the configuration file. The use of a unique directory per pattern enables the deletion of it once the simulation is done based on the “run_testbench_simulations [-keep_simulation_data on | off | on_failure](#)” option value.

- *simulator.simulation_script*

A file that contains simulator-specific commands to map logical libraries to physical directories, compile the test bench as well as the configuration file, and run the simulation.

- *patterns.configuration*

A file that contains a Verilog configuration that is used to control whether a *physical_block* instance uses the full view or the interface view when simulating the given patterns test bench.

- *simulator.simulation_log*

A file that contains the complete transcript that is generated when running the *simulator.simulation_script*. The simulation manager simply redirects the output of the script into this file. The [check_testbench_simulations](#) command parses those files to look for miscompares.

- *simulator.simulation_log_reference*

A file that contains a copy of a previously-generated *simulator.simulation_log* file. This file is manually created by copying the *simulator.simulation_log* file when you are doing simulations with fault insertion.

This file instructs the [check_testbench_simulations](#) command to expect those exact failures in the newly generated *simulator.simulation_log* file. You can also use the -expected_miscompare_count option but this method does not check on which object and in which test step the miscompares actually happened. For example, if you are doing fault insertion to test the memory repair solution, you want to make sure the miscompares are in the memory BIST step prior to repair and not in the one after repair.

The use of the *simulator.simulation_log_reference* file enables you to precisely specify the location of the miscompares. Those files are typically stored in *pattern_name.simulation[_simulation_id]* directories where the *simulation_id* is not null so as not to lose the fault-free simulation directories.

Figure 5-17. Outdir Directory Structure for VCS

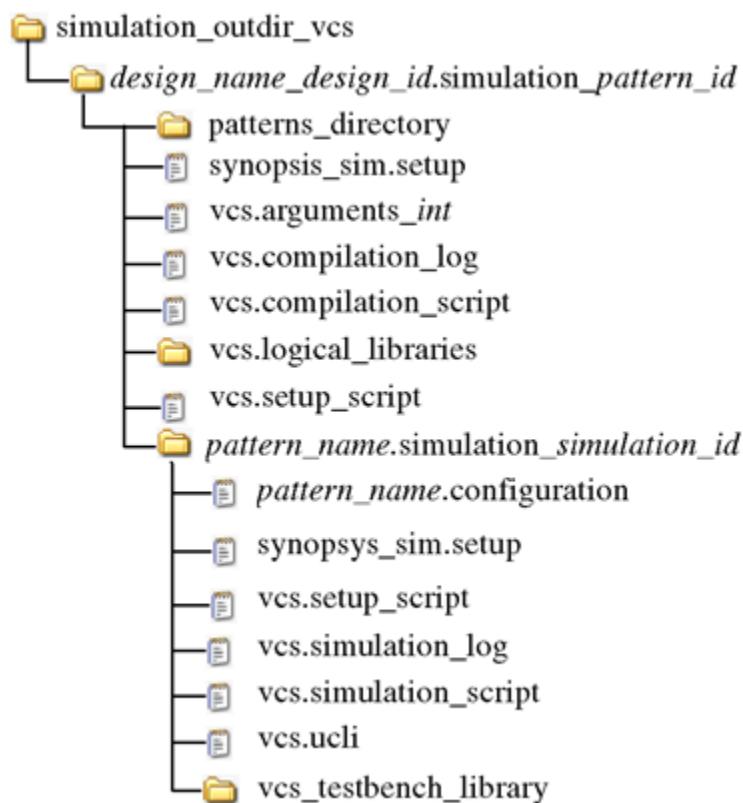
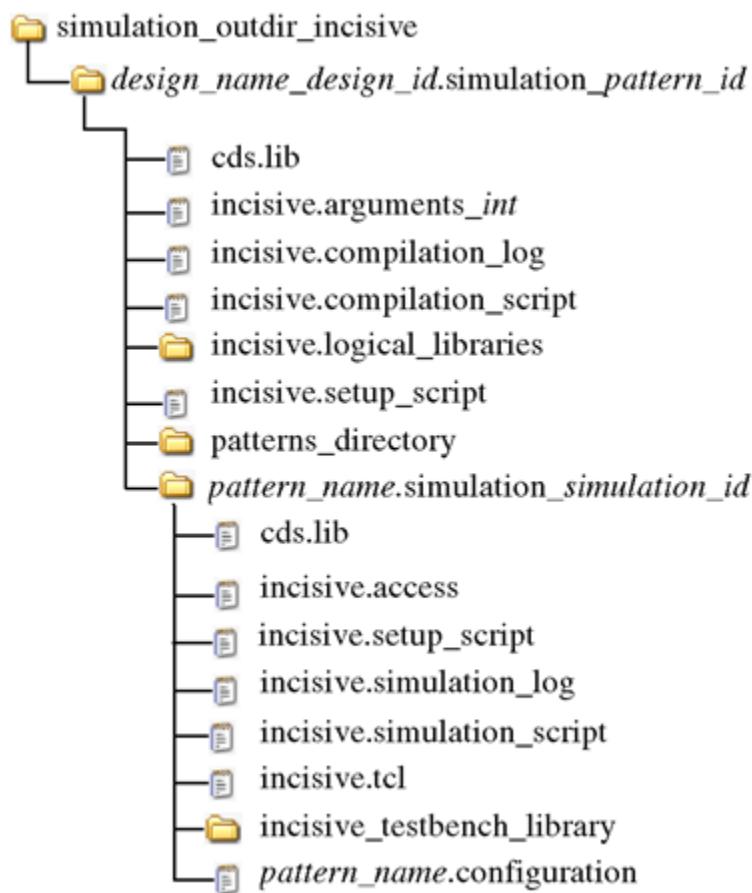


Figure 5-18. Output Directory Structure for Incisive



Environment Variables

Simulation Script Variables

For the `run_testbench_simulations` command, a list of environment variables is exported to a side shell script that is used by the synthesis scripts when they are executed. This way, the execution should be consistent even when executed standalone, without the simulation/synthesis managers.

This is particularly important because the environment variables can be changed in Tesson Shell. When changed this way, the modified environment variables were automatically inherited when sub-invoking a shell command with the '`exec`' or '`system`' command. However, since those environment variable were not exported to the simulation/synthesis scripts, the behavior could be different than when sub-invoking the scripts directly with '`exec`' or '`system`' from Tesson Shell if some key environment variables were modified inside Tesson Shell.

By default, the following list of environment variables are exported:

- PATH
- LD_LIBRARY_PATH

- * _HOME
- * _VERSION
- * _LICENSE_FILE
- MTI* (set to “64” when the MTI_VCO_MODE variable is *not* set)

In addition to the variables in that list, more environment variables can be exported by specifying a list of names ('*' accepted) in an environment variable.

For simulations, the 'TESSENT_SIM_ENVVAR_LIST' environment variable is used.

For example, in Tesson Shell, using:

```
setenv TESSENT_SIM_ENVVAR_LIST [list MY_VARS_* ANOTHER_VAR]
```

will pick-up and export any environment variable with names that start with 'MY_VARS_' and the variable name 'ANOTHER_VAR'. When a variable name in those lists is optionally prefixed by a '-' sign, it won't be exported. The optional '+' sign is accepted and also, it's the default behavior.

A side file is generated that includes the export of all the selected environment variables and that file is sourced by the main simulation scripts. The file is named *simulator.setup_script* and is exported to the following locations:

```
simulation_outdir/design_name_design_id.simulation_simulation_id
```

```
simulation_outdir/design_name_design_id.simulation_simulation_id/
pattern_name.simulation[_pattern_id]
```

Simulator HOME Environment Variables

When setting the following simulator HOME environment variables to the simulator software's *install_home*:

- QUESTA_HOME
- VCS_HOME
- INCISIVE_HOME

the *run_testbench_simulations* command automatically looks in the following locations:

- *install_path/bin*
- *install_path/tools/bin*

to find the required commands. When found, the generated *simulator.setup_script* will have the PATH environment variable automatically appended with the path to the *bin* directory. Setting this environment variable is not required if the PATH environment variable in Tesson Shell already includes the path to the *bin* directory of the required commands.

Simulation Logfile Variable

When the environment variable TESSENT_SIM_MANAGER_ENABLE_LOG is set to the value 1, a log file is produced that contains the information exchanged between Tesson Shell and the simulation manager. By default, the name of the file is './tesson_sim_manager.log' and the path and name of this file can be specified by the environment variable TESSENT_SIM_MANAGER_LOG_FILE_NAME.

By default, the file is overwritten for every Tesson Shell session. It is possible to append to an existing file by changing the file name and specifying the letter 'a' after the file name. The optional letter 'w' reproduces the default behavior.

For example, the following enables the log append to the specified file, if it already exists:

```
setenv TESSENT_SIM_MANAGER_ENABLE_LOG 1  
setenv TESSENT_SIM_MANAGER_LOG_FILE_NAME './my_tesson_sim_manager.log a'
```

Simulation Instance Manager

The TESSENT_SIM_MANAGER_INSTANCES environment variable can be used to control the number of managers. By default, when this environment variable is not set, the tool creates one manager per user per host. This is the same as setting the value to 1.

When you use a value greater than 1, one manager per user per host per Tesson Shell session is started with the manager name being the following:

```
tesson_sim_manager_pid
```

Using this name, it is possible to find the manager in the PID list reported by the OS using the command:

```
ps -F -C tclsh8.5.exe | cat
```

Simulation Socket Manager

The TESSENT_SIM_MANAGER_SOCKET_SELECTION environment variable is used to manage sockets. The default value is "server" which causes the manager to use any socket channel returned by operating system.

The value "auto" means that the tool automatically selects a socket as returned by the operating system and requests that the started manager use it.

Specifying an integer value forces the started manager to use the specified socket, if available.

Simulation Timeout Manager

The TESSENT_SIM_MANAGER_TIMEOUT_SECS environment variable controls how long the tool waits for the socket manager. By default, the command waits for the manager to start and open a socket for 300 sec (5 minutes). You can use this environment variable to change this value.

Simulation Debug

As an alternative to the enable log environment variable, you can use the TESSENT_DEBUG_CONTROL environment variable to generate a logfile.

You set the TESSENT_DEBUG_CONTROL environment variable to the following value:

```
+tessent_sim_manager/log_file
```

or use the following in Tesson Shell:

```
set_debug_control +tessent_sim_manager/log_file
```

It is possible to generate debug information while the managers are starting/running with the following debug controls:

```
+tessent_sim_manager/show_debug_info
```

This information can be of value to Mentor Graphics personnel in the case where a manager refuses to start or does not perform as expected.

Clock Monitoring During Simulation

By default clock monitoring is enabled during simulation.

Note

- ❑ The clock monitoring relies on the design hierarchy being present. If the design has been ungrouped then it may be necessary to turn off the monitoring using the [monitor_internal_clock_pins](#) property in the [SimulationOptions](#) wrapper or use one of the macros shown in the example below when running the simulation.
-

The this monitoring is done by a Verilog side file generated in the same directory as the simulation test benches as follows:

```
tsdb_outdir/patterns/design_name_design_id.patterns_pattern_id/
patterns_name_CLOCK_MONITOR.v
```

The same file can be found with the *patterns_directory* softlink created by the [run_testbench_simulation](#) command under the following:

```
<simulation_outdir>/<design_name>_<design_id>.patterns_<pattern_id>/patterns_directory/
```

For example, for a Patterns(MemoryBist_P1) entry in the [PatternsSpecification](#), the clock monitoring side-file contains the following:

```
//  
=====  
// File : MemoryBist_P1_CLOCK_MONITOR.v  
// Description : Clock Monitor module for pattern MemoryBist_P1  
//  
//  
=====  
// By default the clock monitoring is enabled.  
// To disable clock monitoring define one of the following macros:  
// MGC_DISABLE_CLOCK_MONITOR  
// MGC_DISABLE_CLOCK_MONITOR_MemoryBist_P1  
//  
`define MGC_ENABLE_CLOCK_MONITOR  
`ifdef MGC_DISABLE_CLOCK_MONITOR  
    `undef MGC_ENABLE_CLOCK_MONITOR  
`endif  
`ifndef MGC_DISABLE_CLOCK_MONITOR_MemoryBist_P1  
    `undef MGC_ENABLE_CLOCK_MONITOR  
`endif  
`ifndef MGC_ENABLE_CLOCK_MONITOR  
    `timescale 100fs/10fs  
`endif  
  
To disable all clock monitoring, use:  
-simulation_macro_definitions MGC_DISABLE_CLOCK_MONITOR  
  
To disable clock monitoring only for the Patterns(MemoryBist_P1), use:  
-simulation_macro_definitions MGC_DISABLE_CLOCK_MONITOR_MemoryBist_P1
```

Arguments

- **-design_name *design_name***
An optional switch and value pair that specifies the design name to which the patterns to simulate are associated with. By default, the name of the *current_design* is used when it is set. If you run the command immediately after running the [process_patterns_specification](#) command, you do not need to specify this option.
- **-design_id *design_id***
An optional switch and value pair that specifies the *design_id* to which the patterns to simulate are associated to. By default, the *pattern_id* used by the previously run [process_patterns_specification](#) command is reused automatically; otherwise, the *design_id* value set using the [set_context](#) command is used. If you run the command immediately after running the [process_patterns_specification](#) command, you do not need to specify this option.
- **-pattern_id *pattern_id***
An optional switch and value pair that specifies the pattern id to which the patterns to simulate are associated to. By default, the *pattern_id* used by the previously run

process_patterns_specification command is reused automatically. If you run the command immediately after running the process_patterns_specification command, you do not need to specify this option.

- -report_list

An optional switch that tells the command to simply echo the list of patterns available in the *design_name_design_id.patterns_pattern_id/simulation.data_dictionary* file after it has been filtered using the -select or -exclude switch values.

- -select *pattern_name_glob_list*

An optional switch and value pair that specifies a list of pattern names or pattern name glob patterns to include in the simulation run. The full list of pattern names can be reported using the -report_list option, and is stored in the simulation.data_dictionary file found inside the *design_name_design_id.patterns_pattern_id* directory.

- -exclude *pattern_name_glob_list*

An optional switch and value pair that specifies a list of pattern names or pattern name glob patterns to exclude from the simulation run. The full list of pattern names can be reported using the -report_list option that is stored in the simulation.data_dictionary file found inside the *design_name_design_id.patterns_pattern_id* directory.

- -generate_scripts_only

An optional switch and value pair that tells the command to fully populate the simulation output directory as described in [Figure 5-16](#), but not actually run the compilation and simulation scripts. You can use this option to create the simulation scripts, and then invoke them yourself instead of letting them be managed by the simulation manager.

- -parallel_simulations {1 | 2...MAX_INT | maxcpu}

An optional switch and value pair that specifies to the manager the number of simulations that are allowed to run in parallel. By default, only one simulation is launched at a time. MAX_INT is the maximum integer value that the host machine allows. The maxcpu literal specifies using all available CPUs on the host machine. You can increase the parallelism based on the number of simulator licenses you have and the number of CPUs and memory available on your machine.

- -expected_miscompare_count *int*

An optional switch and value pair that specifies that the set of patterns selected by the -select option have an expected mismatch count. This option is typically used in combination with the -simulation_id and the -simulation_macro_definition switches when doing fault insertion simulations. The fault insertion logic is triggered by the specified “define” macros. They can also be done with explicit forces inside a side module which you load and compile using the -extra_verilog_files option. Using the -simulation_id option allows you to keep a fault free simulation directory while generating new ones for the fault insertion variations.

A more precise mechanism to document expected failures is to copy the *simulator.simulation_log* file to *simulator.simulation_log_reference* once you have inspected and verified it contains the right mismatches. You can also create a directory

beside the specified *simulation_outdir* having the same name with an extra *.reference* extension.

You copy the *simulator.simulation_log* file in the same subdirectory relative to *simulation_outdir.reference* as it exist in the *simulation_outdir* directory. If the *simulator.simulation_log_reference* is not found beside the log file but *simulator.simulation_log* exists inside *simulation_outdir.reference/same_sub_path*, then this file will be used as the reference. The [check_testbench_simulations](#) command will not only verify that the right number of mismatches exists, but that they actually exist on the right objects and in the right test step. For future reruns of the flow, simply pre-populate the *simulator.simulation_log_reference* files inside the directories in which you are doing fault insertion and the [check_testbench_simulations](#) command will automatically use them.

- **-simulation_id string**
An optional switch and value pair that specifies a simulation id string to name the *pattern_name.simulation[_simulation_id]* directory. This is typically used in conjunction with the -simulation_macro_definitions and/or the -extra_verilog_files options to trigger failures in the simulations. For more information, see the -expected_miscompare_count switch description.
- **-simulator questa | vcs | incisive**
An optional switch and value pair that specifies the simulator you want to use for simulation. The default value is “questa” from Mentor Graphics.
- **-simulation_timeout time | unlimited**
An optional switch and value pair that specifies the time out value you want to use when waiting for a simulation to complete. The default value is “unlimited” which means the simulation will be allowed to run for as long as it takes.
- **-keep_simulation_data on | off | on_failure**
An optional switch and value pair that specifies the conditions under which to keep the simulation data after the simulation has completed. The deletion of the *simulator.logical_libraries* directory and the debug output generated when using the -store_simulation_waveforms option is controlled by this option. The default value is “on_failure”.
- **-store_simulation_waveforms on | off**
An optional switch and value pair that specifies to store simulation waveform data so that they can be reviewed if the simulation failed. The deletion of this data is controlled by the -keep_simulation_data option. The default value is “off”.
- **-simulation_run_commands *simulator_commands***
An optional switch and value pair that specifies the commands to use when the -store_simulation_waveform switch is off. If the -store_simulation_waveforms is set to on, then the content from the -waveform_configuration_commands switch is pre-pended to the -simulation_run_commands switch.

- **-waveform_configuration_commands** *simulator_command_list*

An optional switch and value pair that specifies a list of simulator-specific commands to control the resolution of the stored waveform data. When you specify “store_simulation_waveforms on” without the -waveform_configuration_commands option, all waveforms are stored which may dramatically slow down the simulation on a large design.

- **-compilation_macro_definitions** {*macro[=value]* ...}

An optional switch and value pair that specifies a list of macro definitions to pass to the Verilog compilation command inside the *simulator.compilation_script* file which is where your design is compiled. This is typically used to configure your simulation library.

- **-simulation_macro_definitions** {*macro[=value]* ...}

An optional switch and value pair that specifies a list of macro definitions to pass to the Verilog compilation command inside the *simulator.simulation_script* file which is where your test bench and side modules are compiled. You can use this option to trigger special fault insertion code inside your side modules to perform fault insertion simulation. See the -expected_miscompare_count switch description.

- **-compilation_options** *option_list*

An optional switch and value pair that specifies a list of compilation options (for example, +initreg) that modify the compilation script during runtime. The *option_list* uses the following syntax:

```
-compilation_options {[global_options] [verilog verilog_options] [vhdl vhdl_options]}

○ The verilog options are passed only to the command that compiles Verilog source files.

○ The vhdl options are be passed only to the command that compiles the VHDL source files.

○ The global_options can also be expressed as “global {global_options}”.
```

The braces are mandatory in order to identify the options preceded by the language or “global”.

See [Example 4](#).

- **-simulator_options** *option_list*

An optional switch and value pair that specifies a list of options to pass to the simulation command. These are simulator-specific commands and must be consistently specified with the -simulator option.

If you are using ModelSim as your simulator, this switch can be used to pass in the -debugDB option to the tool so the *vsim.dbg* file is created and can be accessed using the

schematic viewer when debugging your waveforms. You must specify this option in conjunction with the -store_simulation_waveform switch as follows:

```
run_testbench_simulation -store_simulation_waveforms on \
-simulator_options "-debugDB"
```

- **-extra_verilog_files** *list_of_files*

An optional switch and value pair that specifies a list of individual Verilog files to compile in parallel to the test bench file. The *list_of_files* are *actual* Verilog files in contrast to a file containing references to Verilog source files. For example:

```
# A single Verilog file:
run_testbench_simulations -extra_verilog_files a.v

# Multiple Verilog files:
run_testbench_simulations -extra_verilog_files { a.v b.v c.v }
```

The files are typically used to contain codes that perform monitoring or fault insertion in the design. You must list the module names compiled within those files that you want to be active in parallel with the test bench using the -extra_top_modules option.

- **-extra_top_modules** *module_name_list*

An optional switch and value pair that specifies a list of modules to run in parallel with the test bench module. These modules are typically used to contain codes that perform monitoring or fault insertion in the design. You must load these modules in files specified with the -extra_verilog_files option.

- **-simulation_output_directory** *simulation_output_directory*

An optional switch and value pair that specifies the name and path of a directory in which to place the simulation directories. See [Figure 5-16](#) for more information about this directory.

- **-simdut_output_directory** *simdut_output_directory*

An optional switch and value pair that is specified with the -simdut_server option to reference the path to the directory that contains the SimDUT setup files that are required for simulation. Refer to “[Simulating Diagnosis of ATPG Patterns \(Non-TSDB Flow\)](#)” in the *Tessent SiliconInsight User’s Manual for Tessent Shell*.

- **-simdut_server**

An optional argument that specifies to run the test benches in Tessent SiliconInsight SimDUT mode. Refer to “[Simulating Desktop, ATE, and ATPG Behavior](#)” in the *Tessent SiliconInsight User’s Manual for Tessent Shell*.

- **-generate_simdut_files_only**

An optional argument that is specified with the -simdut_server option to indicate that Tessent SiliconInsight should generate the SimDUT setup files. By default, this directory is named SIMDUT.simulation and is located in the simdut_outdir directory. Refer to “[Generating the Setup Specification for ATPG \(Non-TSDB Flow\) Desktop Mode](#)” in the *Tessent SiliconInsight User’s Manual for Tessent Shell*.

- **-no_wait**

By default, the `run_testbench_simulations` command automatically waits for the simulations to complete and periodically updating a status line giving the progress of the ongoing simulations.

The optional `-no_wait` switch instructs the command to dispatch the simulation to the manager and not wait for them to complete. It is then possible to follow the progress of the simulations by invoking the `check_testbench_simulations` command. In fact, calling the `run_testbench_simulations` command without the '`-no_wait`' option is equivalent to doing the following:

```
run_testbench_simulations -no_wait ...
check_testbench_simulations -wait
```

Examples

Example 1

The following example runs the `run_testbench_simulations` command twice—once with no options and once to select a few patterns for which fault insertion is performed. Notice the use of the `-simulation_id` in the second invocation which allows the fault-free and fault-injected simulations to co-exist even when the same pattern name is reused. The example assumes that the `simulator.simulation_log` files were copied to `simulator.simulation_log_reference` in the `MemoryBist_*.simulation_FI` directories using the result of a previous run.

```
run_testbench_simulations
run_testbench_simulations -select MemoryBist_* \
    -simulation_macro_definitions {mem32x56_hi_add=4 mem32x56_hi_bit=2} \
    -simulation_id FI
check_testbench_simulations
```

Example 2

The following example shows how to use the “`write_design -tsdb -softlink_netlist`” command and switches to create a TSDB `dft_inserted_designs` container for a netlist generated from synthesis or layout, and on which you want to use the `run_testbench_simulations` command to simulate the patterns described in your [PatternsSpecification](#).

As shown in [Example 3](#) of the `read_design` command, you read in the post-synthesis or post-layout netlist using `read_verilog`, and then you read the ICL and TCD description using the `read_design -no_hdl` command. You then write out a TSDB `dft_inserted_design` container using the “`write_design-tsdb -softlink_netlist`” command and switches. If you are using this command to create a TSDB `dft_inserted_design` container for the post-layout view, you would specify `-design_id` with a name other than “gate,” because “`-design_id gate`” was likely used for the post-synthesis view.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Now that you have a dft_inserted_design container for your netlist, you can either create a new PatternsSpecification using [create_patterns_specification](#) or re-use a pre-existing PatternsSpecification that was potentially modified to signoff the pre-synthesis design view.

```
set_context dft -no_rtl -design_id gate
# This loads the ICL and the interface view of the verilog design to get the full port list of the
# design. No need to read the full netlist to run process_patterns_specification
read_design my_design -view interface
set_current_design my_design

# Option 1: Recreate a brand new PatternsSpecification
create_patterns_specification

# Option 2: re-use pre-existing PatternsSpecification
read_config_data \
    ./tsdb_outdir/patterns/my_design_rtl.patterns_spec_signoff
set_config_value PatternsSpecification(my_design,rtl,signoff) -id <1> gate
process_patterns_specification
run_testbench_simulations

process_patterns_specification
run_testbench_simulations
```

Example 3

The following example shows the tool command sequence to setup the libraries, run the simulation, and check the simulation:

```
set_simulation_library_sources -v adk.v -v ram.v
run_testbench_simulations
check_testbench_simulations
```

Example 4

The following example demonstrates using the -compilation_options switch:

```
run_testbench_simulations -generate_scripts_only -simulator vcs \
    -simulator_options {-kdb -lca -debug_all} \
    -compilation_options \
        { global { -kdb } vhdl { -skip_translate_body } verilog { -timescale=1ns/1ps } }
```

Related Topics

[check_testbench_simulations](#)
[report_testbench_simulation_options](#)
[set_simulation_library_sources](#)

Chapter 6

Command Dictionary (S - Z)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual.

Table 6-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode <u>Uncollapsed</u> Collapsed	A boldface font indicates a required argument.
[]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
<i>Italic</i>	dofile <i>filename</i>	An italic font indicates a user-supplied argument.
{ }	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	Braces enclose arguments to show grouping. Do not enter the braces.
	add_ambiguous_paths <u>{path_name</u> -All} [-Max_paths number]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort <u>ON</u> <u>OFF</u>	An underlined item indicates either the default argument or the default value of an argument.
...	add_clocks <i>off_state</i> <u>pin_list</u> ...	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Command Descriptions

This section describes commands, alphabetically from S through Z, that available in Tessent Shell. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 6-2. Commands S Through Z

Command	Description
select_display_instances	Selects the specified objects in the Flat Schematic and Hierarchical Schematic windows.
set_abort_limit	Specifies the abort limit for the test pattern generator.
set_atpg_fill	Specifies how the tool should fill unspecified bits in ATPG patterns.
set_atpg_limits	Specifies the ATPG process limits at which the tool terminates the ATPG process.
set_atpg_timing	Enables and defines parameters for timing-aware ATPG and fault simulation.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in object_spec.
set_au_analysis	Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.
set_bidi_gating	Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.
set_bist_chain_test	Disables all capture clock activity for the specified number of patterns in the fault simulation during execution of the chain test.
set_bist_debug	Sets up a trace of LFSR(s) value during a pattern's shift cycles.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_bist_trace</code>	Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.
<code>set_boundary_scan_port_options</code>	Sets options relative to boundary scan on ports of the design.
<code>set_bus_handling</code>	Specifies the bus contention results that you desire for the identified buses.
<code>set_bus_simulation</code>	Specifies whether the tool uses global or local bus simulation analysis.
<code>set_bypass_chains</code>	Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.
<code>set_capture_clock</code>	Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.
<code>set_capture_handling</code>	Specifies how the tool globally handles the data capture of state elements that have C3 and C4 rule violations.
<code>set_capture_procedures</code>	Enables or disables the tool's use of a specified subset of named capture procedures when generating test patterns.
<code>set_cell_library_options</code>	Sets the specified options for library parsing and design elaboration.
<code>set_cell_model_mapping</code>	Overrides the non-scan to scan model mapping defined by the tool.
<code>set_chain_test</code>	Specifies binary sequences for chain test.
<code>set_checkpointing_options</code>	Enables checkpointing and specifies the associated options.
<code>set_clock_controller_pins</code>	Specifies the connection information for the clock controller pins.
<code>set_clock_controls</code>	Enables and disables the clock control definitions in the test procedure file.
<code>set_clock_gating</code>	Specifies clock gating cells whose unconnected test-enable ports need to be connected to either the scan enable signal or a specified signal (pin).
<code>set_clock_gating_enable</code>	Associates clocks that control clock gating devices with enable signal drivers.
<code>set_clock_off_simulation</code>	Enables or disables the simulation where all clock primary inputs are at their "off" value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_clock_options</code>	Allows you to specify multiple options for a single clock.
<code>set_clock_restriction</code>	Specifies whether ATPG can create patterns with more than one active capture clock.
<code>set_compactor_connections</code>	Specifies the scan chains for the tool to connect to a particular scan channel output.
<code>set_config_value</code>	Sets the value of an element in the configuration data.
<code>set_contention_check</code>	Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.
<code>set_context</code>	Specifies the current usage context of Tesson Shell.
<code>set_core_instance_parameters</code>	Changes the parameters of instruments previously added with the <code>add_core_instances</code> command.
<code>set_current_design</code>	Specifies the top level of the design for all subsequent commands until reset by another execution of this command.
<code>set_current_edt_block</code>	Directs the tool to apply certain subsequent commands to a particular EDT block only.
<code>set_current_edt_configuration</code>	Sets a particular compression configuration for the tool to use within the current design block.
<code>set_current_mode</code>	Specifies the current test mode in pattern retargeting and core mapping flows.
<code>set_current_silicon_insight_setup</code>	Sets the currently selected setup in the Tesson SiliconInsight Setup Specification.
<code>set_current_simulation_context</code>	Sets the current simulation context.
<code>set_decision_order</code>	Specifies how the ATPG selects gate input or output nodes for evaluation during ATPG.
<code>set_dedicated_wrapper_cell_options</code>	This command controls the inferring of the dedicated wrapper cells on all or the specified primary IOs.
<code>set_defaults_value</code>	Sets the default value of a property in one of the DefaultsSpecification (policy) wrappers.
<code>set_design_include_directories</code>	Specifies one or more directory paths that contain `include files for use by subsequent <code>read_verilog</code> commands.
<code>set_design_level</code>	Specifies the level of the current design.
<code>set_design_macros</code>	Specifies one or more defined macros for use by subsequent commands.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_design_sources</code>	Specifies where the tool should look for the definition of undefined modules in the list of files specified by the <code>read_verilog</code> command.
<code>set_dft_enable_options</code>	Enables or disables the control or observe points.
<code>set_dft_specification_requirements</code>	Specifies the requirements to be checked when the <code>check_design_rules</code> command runs, and those to be included into the DftSpecification when the <code>create_dft_specification</code> command is run.
<code>set_diagnosis_options</code>	Specifies the type of diagnosis performed by the <code>diagnose_failures</code> command.
<code>set_display</code>	Sets the DISPLAY environment variable from the tool's command line.
<code>set_dofile_abort</code>	Specifies whether the tool aborts or continues dofile execution if it detects an error condition.
<code>set_drc_handling</code>	Specifies how the tool handles design rule violations.
<code>set_driver_restriction</code>	Specifies whether the tool allows multiple drivers on buses and multiple active ports on gates.
<code>set_edt_abort_analysis_options</code>	Specifies the number of EDT Aborted (EAB) test cubes on which to perform EAB fault analysis.
<code>set_edt_finder</code>	Controls whether the tool automatically identifies EDT logic and updates scan chain pins during test pattern generation.
<code>set_edt_instances</code>	Specifies the location in which the tool places the EDT logic.
<code>set_edt_mapping</code>	Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.
<code>set_edt_options</code>	Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.
<code>set_edt_pins</code>	By default, all EDT pins are created as dedicated pins with default names and no inversions. The same <code>set_edt_pins</code> command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.
<code>set_edt_power_controller</code>	Generates and inserts a power controller in the compression logic to enable low-power shift.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_external_capture_options</code>	Specifies the number of tester cycles to be used in the capture window of all patterns.
<code>set_external_simulator</code>	Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.
<code>set_fails_report</code>	Specifies whether the design rules checker displays clock rule failures.
<code>set_failure_mapping_options</code>	Specifies filters you can apply when performing reverse mapping of top-level failures to the core in hierarchical designs flows.
<code>set_fault_mode</code>	Specifies whether the fault mode is collapsed or uncollapsed.
<code>set_fault_sampling</code>	Specifies the fault sampling percentage used for circuit evaluation or scan identification.
<code>set_fault_subclass_analyses</code>	Turns on or off the analysis for the specified fault class(es).
<code>set_fault_type</code>	Specifies the fault model type for ATPG.
<code>set_flattener_rule_handling</code>	Specifies how the tool globally handles flattening violations.
<code>set_gate_level</code>	Specifies the hierarchical level of gate reporting and displaying.
<code>set_gate_report</code>	Specifies the information displayed by the <code>report_gates</code> command and in the DFTVisualizer windows.
<code>set_gzip_options</code>	Specifies GNU gzip options to use with the GNU gzip command.
<code>set_icl_extraction_options</code>	Customizes the behavior of ICL extraction.
<code>set_icl_scan_interface_ports</code>	Sets the names of the ports in the specified scan interface previously added by <code>add_icl_scan_interfaces</code> that ICL extraction will create in the new ICL top module.
<code>set_iddq_checks</code>	Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.
<code>set_ijtag_instance_options</code>	Sets options on instances for which there is a matched ICL module.
<code>set_ijtag_retargeting_options</code>	Enables you to configure different aspects of ijtag retargeting.
<code>set_insertion_options</code>	Specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_insert_test_logic_options</code>	Sets new prefix/infix values that will be used when logic is created by the <code>insert_test_logic</code> command.
<code>set_internal_fault</code>	Specifies whether the tool allows faults within or only on the boundary of library models.
<code>set_internal_name</code>	Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.
<code>set_io_mask</code>	Modifies the behavior of bidirectional pins so that their expected values will always be X during test cycles in which the primary input portion of the bidirectional pin is being forced.
<code>set_latch_handling</code>	Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.
<code>set_layout_core_instance</code>	Sets the current layout core instance for inter-scan cell logic reporting with the <code>report_scan_polygons</code> command.
<code>set_lbist_controller_options</code>	Specifies global options to configure the LogicBIST controller.
<code>set_lbist_instances</code>	Specifies the instance in which the LogicBIST controller or single chain mode logic is placed.
<code>set_lbist_pins</code>	Specifies the connection information for LogicBIST controller pins and enables renaming LogicBIST pins.
<code>set_lbist_power_controller_options</code>	Specifies creation of the low-power shift controller for LogicBIST.
<code>set_learn_report</code>	Specifies whether the <code>report_gates</code> command can display the learned behavior for a specific gate.
<code>set_lfsr_seed</code>	Sets the initial value of a PRPG or a MISR.
<code>set_lfsrs</code>	Changes the <code>shift_type</code> and <code>tap_type</code> default setting for the <code>add_lfsrs</code> and <code>add_lfsr_taps</code> commands.
<code>set_license_queue_timeout</code>	Specifies the amount of time that the tool queues for one or more required licenses when not immediately available.
<code>set_list_file</code>	Specifies the name of the list file into which the tool places the pins' logic values during simulation.
<code>set_logfile_handling</code>	Specifies for the tool to direct the transcript information to a file.
<code>set_logical_design_libraries</code>	Defines the set of available logical libraries that can be used by the <code>read_verilog</code> and <code>read_vhdl -in_library</code> option.
<code>set_loop_handling</code>	Specifies how the tool handles feedback networks.

Table 6-2. Commands S Through Z (cont.)

Command	Description
set_lpct_condition_bits	Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.
set_lpct_controller	Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.
set_lpct_instances	Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.
set_lpct_pins	Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.
set_macrotest_options	Prepares a black box macro to allow MacroTest conversion of patterns at the boundaries of that instance.
set_memory_identification_options	Sets options and pattern expressions to validate memory TCD are associated with memory modules to be tested by MemoryBIST.
set_memory_instance_options	Set options on or for memory instances to be used by the check_design_rules and create_dft_specification commands.
set_misr_connections	Describes the connections between the scan chain outputs and the MISR.
set_module_matching_options	Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL module to a design module within ICL extraction.
set_multiple_detection	Enables the multiple detection of faults.
set_multiprocessing_options	Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.
set_net_dominance	Specifies the fault effect of bus contention on tri-state nets.
set_net_resolution	Specifies the behavior of multi-driver nets.
set_number_shifts	Sets the number of shifts for loading or unloading the scan chains.
set_observation_point	Specifies the observation point for random pattern fault simulation.
set_output_masks	Ignores any fault effects that propagate to the primary output pins you select.
set_parallel_load_subchains	Specifies whether the internal parallel access functionality is enabled or disabled.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>set_pathdelay_holdpi</code>	Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.
<code>set_pattern_buffer</code>	Stores run-time pattern data in temporary files in the directory specified.
<code>set_pattern_classification</code>	Specifies the order in which to store test patterns.
<code>set_pattern_filtering</code>	Creates a temporary set of sampled scan patterns.
<code>set_pattern_source</code>	Specifies the source of the test patterns.
<code>set_pattern_type</code>	Specifies the type of pattern(s) used by ATPG simulation.
<code>set_physical_translation</code>	Translates special Verilog characters to SPICE-readable characters that Calibre applications can read.
<code>set_possible_credit</code>	Specifies the percentage of credit that the tool assigns possible-detected faults.
<code>set_power_control</code>	Enables low-power ATPG and sets up a switching and/or capture threshold.
<code>set_power_metrics</code>	Calculates shift and capture power during test pattern generation and simulation.
<code>set_procedure_cycle_checking</code>	Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.
<code>set_procedure_retargeting_options</code>	Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.
<code>set_procfile_name</code>	Specifies a new procedure file for the tool to process at a later time.
<code>set_quick_synthesis_options</code>	This command sets options for quick synthesis.
<code>set_ram_initialization</code>	Specifies whether to initialize RAM and ROM gates that do not have initialization files.
<code>set_random_atpg</code>	Specifies whether the tool uses random patterns during ATPG.
<code>set_random_clocks</code>	Specifies whether Tessent FastScan uses combinational or clock_sequential patterns for random pattern simulation.
<code>set_random_patterns</code>	Specifies the number of random patterns that the tool simulates.

Table 6-2. Commands S Through Z (cont.)

Command	Description
set_read_verilog_options	Sets certain options globally for subsequent read_verilog commands.
set_register_value	Allows you to specify the register value variable outside of Setup mode.
set_relevant_coverage	Determines which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.
set_run_synthesis_options	Specifies the default values for options that can be set for the run_synthesis command.
set_scan_chain_options	Specifies options for managing scan chains within the hybrid TK/LBIST flow.
set_scan_enable	Assigns scan_enable signals to specific scan chains.
set_scan_insertion_options	Specifies the high-level options for the scan insertion process.
set_scan_signals	Specifies the input control signals' drivers used by the insert_test_logic command.
set_screen_display	Specifies whether the tool writes the transcript to the session window.
set_shadow_check	Specifies whether the tool identifies sequential elements as "shadow" elements during scan chain tracing.
set_shift_register_identification	Enables/disables shift register identification. The default is ON.
set_silicon_insight_option	Specifies optional settings for running Tesson SiliconInsight.
set_simulation_library_sources	Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the run_testbench_simulations command.
set_simulation_options	The set_simulation_options command controls the behavior of the tool during pattern generation and simulation.
set_skewed_load	Specifies whether the tool includes a skewed load in the patterns.
set_split_capture_cycle	Controls whether the tool updates simulation data between clock edges.
set_stability_check	Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.

Table 6-2. Commands S Through Z (cont.)

Command	Description
set_static_dft_signal_values	Sets the static DFT signal that exists in the design such that the signal can be used as part of a test_setup.
set_static_learning	Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.
set_system_mode	Specifies the operational state you want the tool to enter.
set_tcl_shell_options	Specifies options for the Tcl shell.
set_test_end_icall	Adds an iCall to the start of the test_end procedure.
set_test_logic	Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.
set_test_point_analysis_options	Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.
set_test_point_insertion_options	Sets parameters related to test point insertion.
set_test_point_types	Specifies the type of test points to insert in the design.
set_test_setup_icall	Adds an iCall to the end of the test_setup procedure.
set_testbench_simulation_options	Specifies the default values for a set of options that exist in the run_testbench_simulations command.
set_tied_signals	Changes the default value for floating pins and floating nets which do not have assigned values.
set_timing_exceptions_handling	Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command.
set_tla_loop_handling	Specifies how to simulate feedback loops that contain transparent latches (TLAs).
set_tool_options	Sets certain options for Tesson Shell.
set_trace_flat_model_options	Modifies the default behavior of the trace_flat_model command.
set_trace_report	Specifies whether the tool displays gates in the scan chain trace.
set_transcript_style	Specifies the style in which the tool transcripts commands.
set_transient_detection	Specifies whether the tool detects all zero width events on the clock lines of state elements.

Table 6-2. Commands S Through Z (cont.)

Command	Description
set_transition_holdpi	Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.
set_tristate_gating	Specifies how tri-state devices are controlled during scan chain shifting.
set_tsdb_output_directory	Sets the TSDB output directory used by the process_dft_specification command.
set_visualizer_logging	Writes the commands entered into the Transcript window to the file specified by the <i>enhanced_dofile</i> argument.
set_visualizer_preferences	Controls a subset of DFTVisualizer preferences for the Flat Schematic and Hierarchical Schematic windows.
set_wrapper_analysis_options	This command sets up parameters for the wrapper cells analysis.
set_write_patterns_options	Creates the subset of ports or vector callbacks to use with the write_patterns command.
set_xbounding_options	Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.
set_xclock_handling	Specifies whether the sequential element model outputs X when any of its clock inputs become X.
set_z_handling	Specifies the simulation handling for high impedance signals on internal and external tri-state nets.
set_zhold_behavior	Specifies whether ZHOLD gates retain their state values.
setenv	Sets a shell environment variable within the tool environment.
shutdown_sid_tester	Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.
simulate_clock_pulses	Pulses one or more clocks within the current simulation context.
simulate_forces	Simulates the queued forces in the current simulation context.
simulate_patterns	Performs simulation by applying the specified pattern source.
sizeof_collection	Returns the number of objects in a collection.
sort_collection	Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

Table 6-2. Commands S Through Z (cont.)

Command	Description
start_silicon_insight	Starts Tessent SiliconInsight and initializes the data structures.
stop_silicon_insight	Stops Tessent SiliconInsight.
system	Passes the specified command to the operating system for execution.
trace_flat_model	Traces the flat model taking into account the current simulation context.
uncompress_layout	Uncompresses a Layout Database.
uniquify_instances	Uniquifies the module definition of the specified instance objects found in obj_spec.
unmark_display_instances	Returns the specified marked instances to their original color. Color marking occurs with the mark_display_instances command.
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.
unregister_static_dft_signal_names	Unregisters previously registered static DFT signal names.
unregister_tcl_command	Unregisters a command that was previously registered using the register_tcl_command.
unselect_display_instances	Unselects the specified objects in the Flat and Hierarchical Schematic window(s).
unsetenv	Unsets a shell environment variable within the tool environment.
update_implication_detections	Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.
verify_patterns	Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.
write_atpg_setup	Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.
write_cell_library	Writes out a single library file of all library files loaded by the read_cell_library command.

Table 6-2. Commands S Through Z (cont.)

Command	Description
<code>write_config_data</code>	Writes the configuration data presently in memory into a file. When using the -wrappers option, the data written to the file can be limited to some specific wrappers.
<code>write_core_description</code>	Writes out the core description corresponding to the current chip level.
<code>write_core_timing_constraints</code>	Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.
<code>write_design</code>	Writes the current design in Verilog netlist format to the specified file.
<code>write_design_import_script</code>	Generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted.
<code>write_design_source_dictionary</code>	Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location.
<code>write_diagnosis</code>	Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.
<code>write_edt_files</code>	Creates the files that implement the EDT logic.
<code>write_failing_paths</code>	Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.
<code>write_failures</code>	Injects faults and/or writes failing pattern results to a file.
<code>write_fault_sites</code>	Writes path definitions or bridge entries from the internal list to a file.
<code>write_faults</code>	Writes fault information from the current fault list to a file.
<code>write_flat_model</code>	Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file.
<code>write_icl</code>	Writes out ICL modules created and/or read in with the <code>read_icl</code> command to the specified file.
<code>write_loops</code>	Writes a list of all the current feedback loops to a file.
<code>write_memory_repair_dictionary</code>	Writes the CompressBisrChain configuration file.
<code>write_modelfile</code>	Writes all internal states for a RAM or ROM gate into the file that you specify.
<code>write_patterns</code>	Saves the current test pattern set to a file in a specified format.
<code>write_primary_inputs</code>	Writes primary inputs to the specified file.

Table 6-2. Commands S Through Z (cont.)

Command	Description
write_primary_outputs	Writes primary outputs to the specified file.
write_procedure_testbench	Writes a test bench for verifying correct operation of internal signals used in procedures.
write_procfile	Writes existing procedure and timing data to the named test procedure file.
write_scan_order	Writes the scan chains into the SCANCHAINS section of the ScanDEF file, where each statement describes a single scan chain. ScanDEF is a subset of the DEF language.
write_scan_setup	Writes out a dofile containing all necessary steps for scan insertion.
write_test_point_dofile	Writes out a dofile containing the test points.
write_tsdb_data	Populates the contents of the Tesson Shell Data Base (TSDB).
write_visualizer_dofile	Writes a dofile containing the commands needed to recreate the current instance and data sets displayed in the DFTVisualizer windows.
write_visualizer_preferences	Writes the current DFTVisualizer preference settings to the specified file.
write_window_contents	Saves the current content of the specified DFTVisualizer window to the specified file.

select_display_instances

Context: all contexts

Mode: setup, analysis

Selects the specified objects in the Flat Schematic and Hierarchical Schematic windows.

Usage

```
select_display_instances {{gate_id# | instance_name | pin_pathname} ... [-ADD] | -ALI}  
[-Display {FLAt_schematic |  
HIErarchical_schematic}]
```

Description

Selects the specified objects in the Flat Schematic and Hierarchical Schematic window(s).

By default, the command first unselects any previously selected gates. To keep previously selected gates selected, include the -Add switch.

Arguments

- ***gate_id#***

A repeatable integer that specifies the gate identification number of a gate to select. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- ***instance_name***

A repeatable string that specifies the name of an instance to select.

- ***pin_pathname***

A repeatable string that specifies the name of a pin associated with a gate you want to select.

- **-ADD**

An optional switch that selects specified objects without first unselecting previously-selected objects.

- **-ALI**

A switch that selects all currently displayed gates.

- **-Display {FLAt_schematic | HIErarchical_schematic}**

An optional switch and literal pair that specifies the window in which this command selects objects. The following literals can be used with the -Display switch:

FLAt_schematic — Selects the specified objects in the Flat Schematic window. This is the default behavior.

HIErarchical_schematic — Selects the specified objects in the Hierarchical Schematic window.

Examples

The following example first selects one object in the Flat Schematic window, then selects two more without unselecting the first:

```
select_display_instances /i$144/q  
select_display_instances /i$142/q /i$141/q -add
```

Related Topics

[open_visualizer](#)

[unselect_display_instances](#)

set_abort_limit

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the abort limit for the test pattern generator.

Usage

```
set_abort_limit comb_abort_limit [seq_abort_limit]
```

Description

Specifies the abort limit for the test pattern generator.

The set_abort_limit command performs slightly differently depending on which tool you are using. Use this command when there are some remaining undetected faults and the test coverage is still too low. By increasing the abort limit, you can allow the tool to detect those remaining undetected faults, and thereby raise the coverage. The following paragraphs describe how the command operates for each tool.

Tip

 If you use the [create_patterns](#) command, you do not need to issue this command. This is because create_patterns automatically determines an appropriate abort limit.

The set_abort_limit command specifies, for combinational and/or clock sequential test generation, the maximum number of attempts the test pattern generator allows before aborting a fault. If the limit is too low, the test pattern generator may abort too many faults and fault coverage could be too low. However, if the limit is too high, it may take too much time to complete the test generation process. The default combinational abort limit value is 30.

During the test generation process for a given fault, the test pattern generator attempts combinational or ram sequential test generation first. If this fails to create a test or prove redundancy, and you specified a non-zero sequential depth by using the [set_pattern_type](#) command, the test pattern generator performs clock sequential test generation. The default abort limit for clock sequential test generation is the same as that for the combinational test generation (30).

Arguments

- ***comb_abort_limit***

A required integer that specifies the maximum number of conflicts for each target fault that the test pattern generator allows during the combinational test generation process. The default is 30.

If you set the combinational abort limit to 0 and the test pattern generator can perform clock sequential test generation, the generator does not perform combinational test generation.

- ***seq_abort_limit***

An optional integer that specifies the maximum number of conflicts for each target fault that the test pattern generator allows during the clock sequential test generation process. The default is the current *comb_abort_limit* default.

If you set the sequential abort limit to 0, the test pattern generator does not perform clock sequential test generation.

Examples

The following example performs an ATPG run, then continues the run with a higher abort limit for the maximum number of allowed conflicts:

```
set_system_mode analysis
create_patterns
set_abort_limit 100
create_patterns
```

Related Topics

[create_patterns](#)
[report_aborted_faults](#)
[report_faults](#)

set_atpg_fill

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies how the tool should fill unspecified bits in ATPG patterns.

Usage

`set_atpg_fill {Random | X | 0 | 1}`

Description

Specifies how the tool should fill unspecified bits in ATPG patterns.

Every pattern generated by ATPG has deterministic bits and unspecified bits. This command allows you to specify that the tool fills those unspecified bits with random values (the default) or X's, 0's, or 1's. This command is useful for getting the test cube that the ATPG generated to test a given fault by setting the fill to X.

Note, this command is only valid in Tessent TestKompress when EDT is set to off with the “[set_edt_options -off](#)” command.

Arguments

- Random | X | 0 | 1

A required literal that instructs the tool how to fill unspecified bits in ATPG patterns.

Examples

The following example fills unspecified bits in ATPG patterns with X's:

`set_atpg_fill X`

[set_atpg_limits](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the ATPG process limits at which the tool terminates the ATPG process.

Usage

```
set_atpg_limits {OFF|{[-Cpu_seconds [OFF|integer]] [-Pattern_count [OFF|integer]] [-Test_coverage [OFF|real | {real integer} ...]]}}
```

Description

Specifies the ATPG process limits at which the tool terminates the ATPG process.

The `set_atpg_limits` command determines the limitations under which the ATPG process operates. Upon invocation of the tool, all the command option limitations are off. If you set any of the limitations, and during an ATPG run the tool reaches one of those limits, the tool terminates the ATPG process. You can use any combination of the three arguments (limitation types). When you specify more than one limitation types, the tool terminates the ATPG process when any of the specified limitation types reaches its limit.

You can check the current settings of the `set_atpg_limits` command by using the [report_environment](#) command.

Arguments

- OFF
An optional literal that turns off *all* ATPG process limits previously set with this command.
- -Cpu_seconds OFF | *integer*
An optional switch and argument pair that specifies the maximum number of CPU seconds that any future ATPG process can consume before the tool terminates the process. The argument choices are as follows:

OFF—A literal specifying that there is no limit to the amount of CPU time the ATPG process consumes during an ATPG process. This is the invocation default.

integer—A positive integer that specifies the maximum number of CPU seconds that the tool can consume during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Pattern_count [OFF | *integer*]
An optional switch and argument pair that specifies the maximum number of test patterns that any future ATPG process can generate before the tool terminates the process. The argument choices are as follows:

OFF—A literal specifying that there is no limit to the number of test patterns the ATPG process generates during an ATPG process. This is the invocation default.

integer — A positive integer that specifies the maximum number of test patterns that the tool can generate during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Test_coverage [OFF | *real* | {*real integer*} ...]

An optional switch and argument pair that specifies the maximum percentage of test coverage that any future ATPG process need reach before the tool terminates the process. The argument choices are as follows:

OFF — A literal specifying the 100 percent test coverage limit during an ATPG process.

The tool terminates the ATPG process when either 100 percent coverage is attained or when the ATPG process has completed. This is the invocation default.

real — A positive real number that specifies the maximum percentage of test coverage that the tool should achieve during an ATPG process (value range is 0 to 100.0).

When the tool reaches the maximum, it terminates the ATPG process.

{*real integer*} — One or more pairs of real and integer numbers that specifies the multiple detect coverage. For example, “-test_coverage 90.5 2” specifies the multiple detect test coverage as 90.5 for multiple detection of 2.

When you specify multiple list pairs, such as “-test_coverage 95 1 85 5,” ATPG stops at 95% coverage in the first pass targeting one detect. Then in the second pass, ATPG stops at 95% or if it completes before 95%. This continues for passes 3 and 4. Pass 5 (targeting 5-detect) stops when ATPG reaches 85% or if it completes before 85%. When you specify multiple list pairs, you cannot associate a higher detect number with a higher test coverage limit. When you specify multiple test coverage limits, the tool terminates the ATPG process only when all limits are met.

Examples

Example 1

The following example sets two of the three limits on the ATPG process and then shows the relevant setup data from the report_environment command:

```
set_atpg_limits -cpu_sec 500 -test_coverage 99.5 -pattern_count 100000
report_environment

...
atpg limits =      500.00 sec 95.50% coverage 100000 patterns
...
```

If the ATPG process reaches either of these two limits, the process terminates. Notice that the information from the report_environment command only shows the settings that are different from the invocation defaults of Off.

Example 2

The following example shows ATPG to target 5-detect with 99% test coverage as first detect limit and 95% multiple detect test coverage limit for the 2-detect to 5-detect:

```
set_multiple_detection -guaranteed_atpg_detections 5
set_atpg_limits -test_coverage 99 1 95 2
report_environment
create_patterns
```

Related Topics

[report_environment](#)

set_atpg_timing

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables and defines parameters for timing-aware ATPG and fault simulation.

Usage

```
set_atpg_timing
  [OFF | ON]
  [[-Clock_waveform {DEFAULT | clock_pin_pathname | clock_gate_id}
    {clock_waveform}...]
   [-DELEte_clock_waveform {ALL | clock_pin_pathname | clock_gate_id}...]
   [-SLack_margin_for_fault_dropping {OFF | slack_margin | slack_margin_percentage%}]
   [-TIMING_CRITICAL timing_margin%]]
```

Description

Enables and defines parameters for timing-aware ATPG and fault simulation.

For complete information, see “[Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- OFF | ON
An optional literal that disables or enables timing-aware ATPG.
- -Clock_waveform
An optional switch that defines a clock waveform for evaluating slack and test metrics.

With the exception of SDQL, timing-aware ATPG uses the test clock frequency for the clock waveform you specify with this switch. The test clock frequency is the ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock).

When you issue the [report_atpg_timing](#) command, the tool reports both the test clock and system clock.

Note

 The timing information from the test procedure file is not used. You must define the clock information for all clocks in the design, even for those not used for ATPG (not used in a named capture procedure).

There are two parameters associated with the -Clock_waveform switch:

DEFAULT | *clock_pin_pathname* | *clock_gate_id*

A string that identifies the clock pin either by pin pathname or gate id. Use the DEFAULT keyword to define a waveform for all clocks that are not explicitly

defined. If you define multiple waveforms for a pin or gate id, the last specified waveform is used.

clock_waveform

Defines the waveform of the clock. The clock waveform will be used for the test clock domain.

The *clock_waveform* includes three floating numbers in the following order, separated by commas:

clock_period, offset, pulse_width

The units are the same as what is specified in the SDF file, which defaults to 1 ns.

- **-DELEte_clock_waveform {ALL | *clock_pin_pathname* | *clock_gate_id*}**

An optional switch and string or literal pair that deletes a clock waveform that was previously defined by the -clock_waveform switch. Use the ALL keyword to delete all explicitly defined clock waveforms. The DEFAULT clock waveform is not deleted by the ALL keyword.

- **-SLack_margin_for_fault_dropping [OFF | *slack_margin* / *slack_margin_percentage%*]**

An optional switch and literal or numerical pair that specifies the fault dropping criteria in addition to the fault detection.

The default setting for this switch is 50%. This means that for faults detected by fault simulation, if the testable minimum slack (Tms) is more than 50% of the actual test slack (Ta), then the fault is dropped. (When ATPG targets a fault, that fault is effectively dropped no matter what fault dropping criteria is in effect.)

OFF

Disables the additional dropping criteria, so the detection by the fault simulation is the same as the regular transition faults.

slack_margin

When a real number “r” is specified without a percentage sign (%), it is treated as the slack margin for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

$$(Ta-Tms) < r$$

Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

slack_margin_percentage%

If a real number “r”, ranged in [0, 100.0], is followed by a percentage sign (%), it is treated as the slack margin percentage for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

$$(Ta-Tms)*100/ Ta < r$$

Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

If set to 100%, faults are dropped regardless of slack.

- **-TIMING_CRITICAL timing_margin%**

An optional switch and numerical pair that selects timing-critical faults. Before running ATPG, the static longest delay path is calculated for each fault. The *timing_margin* is calculated using the following equation:

$$\text{Timing_margin\%} = (\text{static_longest_delay_path}) / (\text{static_longest_delay_path} + \text{slack}) * 100\%$$

If the *timing_margin* for a fault is greater than the specified value, the fault will be selected. For example, assume there are two faults (F1 and F2) with static longest delay paths of 8.5 ns and 9.5 ns, respectively. The clock period is 10 ns, which creates a *timing_margin* of 85% for F1 and 95% for F2. If you specify “-timing_critical 90%”, F2 is selected and F1 is not.

ATPG still runs with the complete fault list, but the *slack_margin_for_fault_dropping* is applied only to the selected timing-critical faults. Selecting a large *timing_margin* (90%) is recommended to speed up ATPG. If you want to target only selected timing-critical faults, follow the command sequence described in “[Run Time Reduction for Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual*.

Examples

The following example enables timing-aware ATPG and defines the clock information:

```
set_atpg_timing -clock test_clk 16000 8000 8000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
```

Related Topics

[report_atpg_timing](#)
[report_statistics](#)

set_attribute_options

Context: unspecified, all contexts

Mode: all modes

Modifies options of any registered user-defined attribute.

Usage

```
set_attribute_options -name attribute_name_list [-object_types type_list]
    [-export_during_write {auto | off | on}]
    [-preserve_boundary_in_flat_model {off | on}]
    [-applies_to_child_instances {off | on}]
    [-display_in_gui {off | on}]
    [-gui_marking_index marking_index] [-silent]
```

Description

Modifies options of any registered user-defined attribute.

Specifically, this command modifies the `-applies_to_child_instances`, `-preserve_boundary_in_flat_model`, `-display_in_gui`, and `-gui_marking_index` options of any registered user-defined attribute.

Arguments

- **-name attribute_name_list**

A required switch and value pair that specifies an attribute name or a Tcl list of attribute names whose options are to be modified.

- **-object_types type_list**

An optional switch and value pair that specifies the object type or a Tcl list of design object types whose attribute configuration settings are to be modified. The attributes of object types not included in this list remain unaffected. If this argument is omitted, by default, the attribute options of all design object types for which the attributes are registered on are modified.

- **-export_during_write auto | off | on**

An optional switch and literal pair that specifies which module and port ICL attributes are written into the ICL output when a module is written out using the [write_icl](#) command. Any attribute that does not have this value set is written out in the same form as it was read in; any attribute value set with the [set_attribute_value](#) command since being read in is ignored.

You should use this default behavior when you want to write out the attribute as it was read in. You can modify the default behavior using the following literals:

auto — Use *auto* when you want to update the value of an attribute that was read in with the ICL source. In this case, the value read in from ICL is written out unless an alternative value has been set using the [set_attribute_value](#) command, which will be written out instead. If the ICL source did not have an attribute definition, the value set

using the `set_attribute_value` command is ignored and not written out when the ICL is written.

on — Use *on* when you want to update an attribute that was read in with the ICL source, or to cause a new ICL attribute definition to be created for that attribute when the ICL is written. The attribute value set using the `set_attribute_value` command is written out when the ICL is written out. However, the attribute definitions that were not read in from the ICL source are also written out.

off — Use *off* when you want an attribute to be ignored and omitted as the ICL is being written out. The value read in from ICL and any subsequent attribute value set using the `set_attribute_value` is ignored when the ICL is written out. The output ICL file will not have an attribute definition for this attribute.

- `-preserve_boundary_in_flat_model off` **on**

An optional switch that specifies whether to preserve certain hierarchical pins or nets and their attributes during design flattening based on an attribute of a module, instance, port, pin, or net. The `-preserve_boundary_in_flat_model` only works with hierarchical design objects: modules, instances, ports, pins, and nets. If you attempt to use this option with objects other than hierarchical design objects, then the tool issues a warning and ignores the option.

For module attributes, this option preserves all pins of all instances of the module, if the attribute is set to a non-default value. For instance attributes, this option preserves all pins of the instance, if the attribute is set to a non-default value. For port attributes, this option preserves only the corresponding pin on each instance of the port's module, if the attribute is set to a non-default value. These preserved hierarchical pins become buffers in the flat model and are no-faulted. By default, hierarchical pins are not preserved. For net attributes, this option preserves the net boundary with a buffer in the flat model which drives the original fanout of the net and provides a [Gate_pin](#) having the name of the net.

If you set an attribute on a pin of a design model, instance, port, pin, or net by enabling this option, the flat model will need to be reconstructed. The attribute is not lost during flattening.

Note that by default, only the boundaries of library cells are preserved in the flat model. (Those are the locations where faults are added for ATPG, for example.) So the boundaries of higher-level modules in the design are not preserved in the flat model by default. When you use the “`-preserve_boundary_in_flat_model on`” switch, each hierarchical pin or net and boundary of an instance is preserved in the flat model with place-holder buffers corresponding to each pin or net. A `gate_pin` object then exists in the flat model to match the hierarchical pin or net. The instance names of the pin or net and `gate_pin` are the same.

- `-applies_to_child_instances off` **on**

An optional switch that defines if the attribute is also visible on the instances below the instances or the instances of the modules on which the attribute is defined. This option can only be set to true on attributes defined for objects of type instance or module.

- **-display_in_gui {off|on}**

An optional switch and literal pair that specifies whether the attribute(s) specified by the name argument is visible in the Debug and DesignFlat and Hierarchical Schematic windows of DFTVisualizer.

off — The attribute is not visible in the Debug and DesignFlat and Hierarchical Schematic windows.

on — The attribute is visible in the Debug and DesignFlat and Hierarchical Schematic windows.

- **-gui_marking_index *marking_index***

An optional switch and value pair that specifies the marking index that DFTVisualizer uses when displaying the attribute(s). The marking index is an integer (1-10) that maps to one of ten colors. DFTVisualizer uses the specified color as a background when it displays the attribute. The default is 1 which maps to light blue.

- **-silent**

An optional switch that suppresses error messages if one or more attributes in *attribute_name_list* do not exist.

Examples

Example 1

The following example shows how to configure the attribute named power_domain in order for it to be visible in all child instances of the instance for which it is defined:

```
set_attribute_options -name power_domain -object_types instance
                     -applies_to_child_instances on
```

Example 2

This example demonstrates how an attribute value is written or not written into the ICL module representation based on the value of the set_attribute_options -export_during_write setting. This example assumes that the following ICL module has been read in:

```
Module block {
    DataInPort din;
    DataOutPort dout;
    Attribute speed = "100";
}
```

Executing the following commands in Tessent Shell, using the default set_attribute_options settings, produces the module definition that follows:

```
register_attribute -name speed -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name speed -value "200"
register_attribute -name added_by -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name added_by -value "John"

# report the ICL module with the default attribute export_during_write setting
report_icl_module block
```

```
Module block {
    DataInPort din;
    DataOutPort dout;
    Attribute speed = "100";
}
```

Executing the following additional commands in Tessent Shell, using the `set_attribute_options auto` setting, produces the module definition that follows:

```
# set the attribute to auto export
set_attribute_options -name speed -export_during_write auto
set_attribute_options -name added_by -export_during_write auto
report_icl_module block

Module block {
    DataInPort din;
    DataOutPort dout;
    Attribute speed = "200";
}
```

Executing the following additional commands in Tessent Shell, using the `set_attribute_options on` setting, produces the module definition that follows:

```
# set the attribute to always export
set_attribute_options -name speed -export_during_write on
set_attribute_options -name added_by -export_during_write on
report_icl_module block

Module block {
    DataInPort din;
    DataOutPort dout;
    Attribute speed = "200";
    Attribute added_by = "John";
}
```

Executing the following additional commands in Tessent Shell, using the `set_attribute_options off` setting, produces the module definition that follows:

```
# set the attribute to always export
set_attribute_options -name speed -export_during_write off
set_attribute_options -name added_by -export_during_write off
report_icl_module block

Module block {
    DataInPort din;
    DataOutPort dout;
}
```

Related Topics

[get_attribute_list](#)
[get_attribute_option](#)

[get_attribute_value_list](#)
[register_attribute](#)
[report_attributes](#)
[reset_attribute_value](#)
[set_attribute_value](#)
[unregister_attribute](#)

set_attribute_value

Context: unspecified, all contexts

Mode: all modes

Sets an attribute's value for the objects specified in *object_spec*.

Usage

```
set_attribute_value object_spec -name attribute_name -value attribute_value [-silent]
```

Description

Sets an attribute's value for the objects specified in *object_spec*. This command returns a collection of objects on which the attribute value has been set.

If the command fails setting the attribute on any element, an error message is given to the user and the command returns TCL_ERROR. This can be suppressed by using -silent (No error messages and no TCL_ERROR).

[Table 6-3](#) lists some of the attributes that you can specify with the `set_attribute_value` command.

Table 6-3. User-Specifiable Attributes

Name	Definition	Object	Values
default_fault_classification_0	<p>String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-0 or slow-to-rise. You use the “set_attribute_value” command to set the desired default fault classification. During the flow, when the “add_faults” and “load_faults” commands are called, or when the tool internally adds faults for scan-based test pattern generation, the fault sites marked with this attribute will be initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it will be ignored.</p> <p>This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test.</p> <p>See Example 4 below for a typical use model</p>	pin or gate_pin	Any pre-defined fault class and subclass(if fault class is AU or DI). Default value is UC.

Table 6-3. User-Specifiable Attributes (cont.)

Name	Definition	Object	Values
default_fault_classification_1	<p>String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-1 or slow-to-fall. You use the “set_attribute_value” command to set the desired default fault classification. During the flow, when the “add_faults” and “load_faults” commands are called, or when the tool internally adds faults for scan-based test generation, the fault sites marked with this attribute will be initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it will be ignored.</p> <p>This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbrisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test.</p> <p>See Example 4 below for a typical use model</p>	pin or gate_pin	Any pre-defined fault class and subclass(if fault class is AU or DI). Default value is UC.
ignore_for_graybox	Specifies a primary output/bidi pin to exclude from graybox analysis. By default, this attribute is set to false for all primary output/bidi pins, hence they are considered by the graybox analysis as backtracing origins. Setting this attribute on an output/bidi pin of a sub-module or an input pin of any module has no effect on graybox analysis. You need to exclude the scan output pins of core chains (or channel output pins if it is a design containing EDT hardware) from graybox analysis using this attribute because the tool cannot automatically determine which output/bidi pins are for core chains. For more information about this attribute, refer to the analyze_graybox command description.	pin	true, false (default)
in_graybox	Specifies instances to include or exclude from graybox netlist. For more information about this attribute, refer to the analyze_graybox command description.	instance, net	true, false (default)

Table 6-3. User-Specifiable Attributes (cont.)

Name	Definition	Object	Values
is_hard_module	When true, prevents a module from having any changes to its internal logic during test logic insertion, meaning that the module is written to the scanDEF file unexpanded (the scanDEF file references the hard macro's module instance rather than any flip-flops inside).	module	true, false (default)

Arguments

- ***object_spec***
A required value that specifies a Tcl list of one or more object names, or a collection of one or more objects.
- **-name *attribute_name***
A required switch and value pair that specifies the name of the attribute to set.
When an initial fault classification for a pin is specified with the default_fault_classification_0 or default_fault_classification_1 attributes (for example, DI.USER), the fault will first be analyzed through DI/UU/TI/BL fault analysis and will only be added by its specified name (DI.USER) if it is not classified as a DI/UU/TI/BL fault.
- **-value *attribute_value***
A required switch and value pair that specifies the value to which to set the attribute.
- **-silent**
An optional switch that suppresses error messages.

Examples

Example 1

The following example sets the is_hard_module attribute to true for all modules whose names starts with mem_.

```
set_attribute_value [get_modules mem_*] -name is_hard_module
{mem_a mem_b mem_c}
```

Example 2

This example sets the user-defined attribute clock_domain to clka on pin u1/u2/ck.

```
set_attribute_value u1/u2/ck -name clock_domain -value clka
{u1/u2/ck}
```

Example 3

The following example excludes clk1 and clk2 from X-constraining during core-level retargetable pattern generation of the CPU core.

```

SETUP> set_context patterns -scan
SETUP> set_current_mode -type internal
SETUP> read_verilog CPU.v
SETUP> read_cell_library atpglib
SETUP> set_current_design
SETUP> set_attribute_value {clk1 clk2} -name \
    is_excluded_from_isolation_constraints
SETUP> set_system_mode analysis
ANALYSIS> write_core_description CPU.tcd -replace
ANALYSIS> create_patterns
ANALYSIS> write_patterns CPU_stuck.retpat.qz -replace
ANALYSIS> write_faults CPU_stuck.faults.qz -replace
...

```

Example 4

This example uses the trace_flat_model command to find all the fault sites in the combinational fanout of a port and initialize them as “DI.my_test_mode”. This port is not testable by scan-based test generation, but it is covered by another test mode called “my_test_mode”. Instead of ending up as AU, the fault sites are marked as DI.my_test_mode.

```

SETUP> set_context patterns
SETUP> read_cell_library atpgff.atpg_lib
SETUP> read_verilog design.v
SETUP> set_current_design
SETUP> create_flat_model
SETUP> set gate_pins [trace_flat_model \
    -from my_test_port \
    -direction forward \
    -controllability connected \
    -tag_condition "fault_site"]
SETUP> set_attribute_value $gate_pins -name default_fault_classification_0 -value
DI.my_test_mode
SETUP> set_attribute_value $gate_pins -name default_fault_classification_1 -value
DI.my_test_mode

```

Related Topics

- [get_attribute_list](#)
- [get_attribute_option](#)
- [get_attribute_value_list](#)
- [register_attribute](#)
- [reset_attribute_value](#)
- [set_attribute_options](#)
- [report_attributes](#)

[set_au_analysis](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.

Usage

`set_au_analysis {ON | OFF}`

Description

Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.

The `set_au_analysis` command specifies whether the ATPG process can use the ATPG untestable information. Upon invocation of the tool, the AU analysis is set to On; therefore, the ATPG process uses the ATPG untestable information. Once the tool places a fault in the AU fault class, it removes the fault from the active fault list and does not simulate it. This prevents the ATPG process from identifying the faults as possibly-detected during an ATPG run. However, you may use a switch on the `compress_patterns` command to identify possible detections of AU faults.

Arguments

- **ON**

A literal that specifies for the tool to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes. This is the invocation default.

- **OFF**

A literal that specifies for the tool not to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes.

Examples

The following example specifies not to use ATPG untestable information during the ATPG run:

```
set_system_mode analysis
set_au_analysis off
create_patterns
```

Related Topics

[compress_patterns](#)

[delete_atpg_constraints](#)

[read_faults](#)

set_bidi_gating

Context: dft -scan, dft -test_points

Mode: setup

Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.

Usage

```
set_bidi_gating [Off | ON | Scan] [-Control {SEn | TEn}] [-Direction Input | Output]
[-Top {All | primary_bidi_pin...}] [-Force_gating]
```

Description

Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.

When enabled, test logic is inserted for bidi pins as necessary to control the enable signal and/or the input/output direction as specified.

By default, when the enable signal of a bidi pin is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

You can also specify which enable signal (TEN or SEN) enables bidi pins.

Arguments

- **Off | ON | Scan**

Required literal that specifies whether to insert test logic to control the enable lines of bidi pins during scan chain shifting. For more information on scan chain shifting, see “[Test Logic Insertion](#)” in the *Tessent Scan and ATPG User’s Manual*. Literal options include:

Off — no test logic is inserted to control bidi pins. Default setting.

ON — test logic is inserted as necessary to control bidi pins.

Scan — inserts test logic on the scan I/O bidi pins to control the direction of the bidi pin for scan shifting and ensure the success of scan chain tracing. Scan output bidi pins are gated to be in output mode while all other bidi pins are gated to be in input mode (Z state on the tester).

- **-Control SEn | TEn**

An optional switch and literal pair that specifies the enable signal used to control bidi pins. Options include:

SEn — scan_enable signal. Default setting.

TEn — test_enable signal.

- **-Direction Input | Output**

An optional switch and literal pair that specifies the direction of the bidi pins specified by the **-Top** switch. Options include:

Input — bidi pins are gated so they are in input mode. Default setting.

Output — bidi pins are gated so they are in output mode.

- **-Top All | *primary_bidi_pin*...**

An optional switch and literal or repeatable string pair that specifies which bidi pins are controlled with the specified enable signal. Options include:

All — all bidi pins. Default setting.

primary_bidi_pin — a specified primary bidi pin. Test logic is inserted to ensure that these bidi pins are controlled as specified.

- **-Force_gating**

An optional switch that adds test logic to the enable lines of bidirectional pins that are directly controlled by primary inputs, by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Examples

Example 1

The following example uses the **set_bidi_gating** command to insert test logic to control all bidi pins used for scan I/O via the SEN signal, and reports the gated bidi pin.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
set_system_mode analysis
add_scan_mode unwrapped -si_connections c1 bidi_in1/X \
    -so_connections blkB1/blkA/utri2/A
analyze_scan_chains
```

Example 2

The following example uses the **force_gating** switch to insert gating logic controlled by the TEN control signal on the enable line of /ui01 and reports the gated tri-state devices. /ui01 is a bidirectional device driving primary inout port dinout[1]; its enable signal is directly controlled by the primary input /io_control1.

```
set_bidi_gating on -control ten -direction input -top dinout[1] -force_gating
```

Related Topics

[report_test_logic](#)

[report_control_signals](#)

[set_test_logic](#)

[set_tristate_gating](#)

set_bist_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Disables all capture clock activity for the specified number of patterns in the fault simulation during execution of the chain test.

Usage

`set_bist_chain_test [None | number_of_patterns]`

Description

Disables all capture clock activity for the specified number of patterns in the fault simulation during execution of the chain test.

By default, the tool runs a 32 pattern chain test and disables all capture clock activity for those 32 patterns, which means that the unload values will exactly match the load values during those patterns.

Hybrid TK/LBIST Flow

The tool automatically includes this command and literal in the generated logicBIST fault simulation dofile—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- None
An optional literal that specifies that the tool should not disable the capture clock activity since there is no chain test being ran.
- *number_of_patterns*
An optional integer that specifies for the tool to disable all capture clock activity for the number of chain test patterns that are to be driven through the scan chains, which is 32 by default.

Related Topics

[read_patterns](#)

[set_random_patterns](#)

set_bist_debug

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Sets up a trace of LFSR(s) value during a pattern's shift cycles.

Usage

```
set_bist_debug -None | [LFSR_name... {pattern_number | All}]
```

Hybrid TK/LBIST Flow Usage

```
set_bist_debug -None | [LFSR_name... {pattern_number | All}] |  
[-decompressor [block_name] {pattern_number | All}]
```

Description

Sets up a trace of LFSR(s) value during a pattern's shift cycles.

To facilitate signature mismatch debugging and other diagnostics, the set_bist_debug command enables the tracing of all specified MISR and PRPG values associated with a given pattern, or all BIST patterns. If low-power hardware is enabled in the hybrid TK/LBIST flow, the low-power hold register value is traced along with its corresponding PRPG.

Arguments

- **-None**

A switch that clears the list of PRPGs and MISRs the tool is currently monitoring.

- ***LFSR_name***

A repeatable string that specifies the name of the MISRs and PRPGs whose values the tool will trace. To list all MISRs and PRPGs in the design, use the report_lfsrs command.

Of the next two arguments, you must choose one.

- ***pattern_number***

An integer that specifies the pattern number for which the MISR or decompressor values will be traced.

- **All**

A literal that specifies to write out the value of the specified MISRs and PRPGs only at the end of the pattern, for every pattern.

Hybrid TK/LBIST Flow Arguments

- **-decompressor**

A required switch that takes an optional *block_name* to specify that decompressor values for this block should be reported. (The decompressor is acting as a PRPG during logicBIST.) When using a single EDT module, the *block_name* is optional but is required in the modular

EDT flow. If you do not specify *block_name*, the tool reports decompressor values for all blocks.

- *block_name*

An optional string that specifies the name of the EDT block containing the decompressor to evaluate. The *block_name* must be one you defined previously using the [add_edt_blocks](#) command.

Examples

Example 1

The following example initiates a trace on the values of both the MISR and the PRPG for the pattern, 61, and shows the resulting transcript:

```
// command: set_bist_debug misr prpg 61
// command: simulate_patterns
// -----
// Simulation performed for #gates = 5732 #faults = 418
// system mode = fault simulation      pattern source = 63 BIST patterns
// -----
// #patterns test      #faults #faults    # eff.    # test      process
// simulated coverage in list detected   patterns patterns  CPU time
// begin bist patterns: capture clock = /clk, observe point = MASTER
begin debug data for PRPGs pattern = 61
    PRPG      prpg    0  100000111111010
    PRPG      prpg    1  010000011111101
    PRPG      prpg    2  100101001111110
    PRPG      prpg    3  010010100111111
    .
    .
    .
    PRPG      prpg    14 010101110010010
    PRPG      prpg    15 001010111001001
    PRPG      prpg    16 1010000111100100
// 63      88.68%    387     31      5      0      0.04 sec
begin debug data for MISRs pattern = 61
    MISR      misr    1  111011010010011
    MISR      misr    2  011101101011110
    MISR      misr    3  001110110011111
    MISR      misr    4  000111011111111
    .
    .
    .
    MISR      misr    16 1100100000101111
    MISR      misr    17 1100100000101111
```

Example 2

The following example initiates a trace on the values of the PRPG and MISR for all patterns:

```
// command: set_bist_debug misr prpg all

// command: simulate_patterns
// -----
// Simulation performed for #gates = 5912 #faults = 513
// system mode = fault simulation pattern source = 63 BIST patterns
// -----
// #patterns test #faults #faults # eff. # test process
// simulated coverage in list detected patterns patterns CPU time
// begin bist patterns: capture clock = /clk, observe point = MASTER
PRPG      prpg    0  0001001000001001
PRPG      prpg    1  0011001000011010
PRPG      prpg    2  001011111001111
PRPG      prpg    3  0110011101111011
.
.
.
PRPG      prpg    29 0001101111101000
PRPG      prpg    30 0100010010110000
PRPG      prpg    31 1110101100101111
MISR     misr    0 0101010100001101
MISR     misr    1 1101011101111111
MISR     misr    2 1101010001101001
MISR     misr    3 1110000010111001
.
.
.
MISR     misr    29 0010001111001110
MISR     misr    30 1111000010000110
MISR     misr    31 1111001001110101
PRPG      prpg    32 1000110000011000
PRPG      prpg    33 0101010011111101
PRPG      prpg    34 0100001010111010
.
.
.
PRPG      prpg    60 100000111111010
PRPG      prpg    61 1010000111100100
PRPG      prpg    62 0000000110101110
// 63 86.05% 488   25   3   0   0.10 sec
MISR     misr    32 1000101100100001
MISR     misr    33 0010101011010011
MISR     misr    34 1001100110111101
.
.
.
MISR     misr    60 1011110010101001
MISR     misr    61 0011100101011010
MISR     misr    62 0001111111111100
```

Example 3

The following example initiates the trace on the values of the decompressor, which is used as a PRPG in the hybrid TK/LBIST flow, for pattern 4 and shows the resulting transcript. In this example, low power is enabled.

```
// command: set bist debug -decompressor 4

begin debug data for DECOMPRESSORs pattern = 4
DECOMP picccpu_edt_i.lfsm_vec 0 100111001001100101010011001001001
LP picccpu_edt_i.lfsm_vec 0 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 1 0100111001001000101010011001001
LP picccpu_edt_i.lfsm_vec 1 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 2 1010011101100100010101001100100
LP picccpu_edt_i.lfsm_vec 2 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 3 0101011110110110001010100110010
LP picccpu_edt_i.lfsm_vec 3 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 4 0010111110011011000101010011001
LP picccpu_edt_i.lfsm_vec 4 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 5 1001011111001001100010101001100
LP picccpu_edt_i.lfsm_vec 5 0100110000000110010010001100110
DECOMP picccpu_edt_i.lfsm_vec 6 0100101110100100110001010100110
LP picccpu_edt_i.lfsm_vec 6 0100110000000110010010001100110
.
.
.
```

Related Topics

[set_bist_trace](#)
[add_lfsrs](#)
[report_lfsrs](#)

set_bist_trace

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.

Usage

```
set_bist_trace [-Nolfsr | {-Lfsr filename [-Replace] [-Interval interval] [-Offset offset]})]
```

Hybrid TK/LBIST Flow Usage

```
set_bist_trace [-prpg_data_file filename -misr_data_file filename]
  {[-prpg_pload_file filename -misr_pload_file filename] [-range beg_int end_int]}
  [-low_power] [-replace]
```

Description

Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.

The traced values are logged in *filename* in the following format:

```

0 PRPG prpg 1111111111111111
0 MISR misr 1111111111111111
1 PRPG prpg 1001010001011000
2 PRPG prpg 0101111001100101
...
32 PRPG prpg 1100100000000110
1 MISR misr 11100111011000010110
2 MISR misr 11001110011100001110
3 MISR misr 01011001100111011001
...
32 MISR misr 01011001100111011001
32 PRPG prpg 1100100000000110
...

```

The first element on each line is the pattern number, followed by the simulated element type (MISR or PRPG). The third element is the name of the simulated element. The names shown in the previous example are the default names assigned by the add_lfsrs commands included in the BIST-Controller-phase-generated fault simulation driver. You can edit the driver file to rename the elements.

Note

 Do not modify or delete any header information in *filename*. Microcoded test bench instructions are generated based on information in this header.

The sequence of activities that occur within the simulation of the BIST controller determines the order of the lines in *filename*.

If you issue multiple `set_bist_trace` commands before the [simulate_patterns](#) command, only the last one issued is executed.

To reduce the volume of data written to *filename*, use the `-Interval` and `-Offset` switches. See the [Example 1](#) section for this command.

Arguments

- **`-Nolfsr`**
An optional switch that disables LFSR value tracing. This is the default.
- **`-Lfsr filename`**
An optional switch and string pair that enables tracing of PRPG and MISR values during a BIST run and writes those values to the *filename* file.
- **`-Replace`**
An optional switch that replaces the contents of *filename* if that file already exists.
- **`-Interval interval`**
An optional switch and integer pair that writes the PRPG and MISR values to *filename* for the patterns beginning at 0 and separated by the *interval* given by the *interval* integer. The values for the first and last patterns are always written.
- **`-Offset offset`**
An optional switch and integer pair that writes the PRPG and MISR values to *filename* beginning at the pattern number given by the *offset* integer. The values for the first and last patterns are always written.

Hybrid TK/LBIST Flow Arguments

- **`-prpg_data_file filename -misr_data_file filename`**
Two optional switches and string pairs that enable tracing of the PRPG and MISR values for all patterns during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow.
- **`-prpg_pload_file filename -misr_pload_file filename`**
Two optional switches and string pairs that enable tracing for a few important clock cycles of the PRPG and MISR values during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow and are used for parallel load simulations.
- **`-range beg_int end_int`**
An optional switch and integer pair that defines a range of patterns to be written out to the PRPG/MISR files when using the `-prpg_pload_file` and `-misr_pload_file` options. By default, the full pattern range for a BIST run is written out.

- **-low_power**

An optional switch that defines whether the hybrid TK/LBIST flow should have the low-power values written out to the PRPG file when using the **-prpg_data_file** and **-prpg_pload_file** options.

- **-replace**

An optional switch that replaces the contents of *filename* if that file already exists.

Examples

Example 1

The following example writes traced values to the *lfsr.trace* file, but only writes the values for every 256th pattern to conserve disk space.

```
set_bist_trace -lfsr lfsr.trace -replace -interval 256
```

In the preceding example, if you discover a problem begins after pattern 1530, you might narrow the search by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1530 -interval 16
```

and then finish by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1594
```

Example 2 (Hybrid TK/LBIST Flow)

The following example writes out PRPG and MISR data files, with low-power values:

```
set_bist_trace -prpg_data_file LV_WORKDIR/TOP.prpg_data_lbist \
-misr_data_file LV_WORKDIR/TOP.misr_data_lbist \
-low_power \
-replace
```

Example 3 (Hybrid TK/LBIST Flow)

The following example writes out the PRPG and MISR parallel load data files, with low-power values, for the first 256 patterns:

```
set_bist_trace -prpg_pload_file LV_WORKDIR/TOP.prpg_pload_lbist \
-misr_pload_file LV_WORKDIR/TOP.misr_pload_lbist \
-low_power \
-range 0 255 \
-replace
```

Related Topics

[set_bist_debug](#)

set_boundary_scan_port_options

Context: dft

Mode: setup

Sets options relative to boundary scan on ports of the design.

Usage

```
set_boundary_scan_port_options {port_spec -cell_options cell_options} |  
    -pin_order_file file_name | -pad_io_ports pad_io_ports
```

Description

Sets options relative to boundary scan on ports of the design.

The -pin_order_file and -pad_io_ports options apply to all ports.

When the [set_dft_specification_requirements](#) -boundary_scan option is set to on, the options defined with this command are considered by the [check_design_rules](#) and [create_dft_specification](#) commands.

Arguments

- *port_spec*

An optional string that specifies a Tcl list of names of one or more ports, or a collection containing one or more ports of the current design.

- -cell_options *cell_options*

An optional literal pair that defines the options to associate to the specified ports. The allowed values are:

```
normal | no_bscan_cell | dont_touch | no_connect | analog |  
compliance_enable0 | compliance_enable1 |  
{output_only no_capture_core_signal} | {input_only {clock |  
sample_only}} | add_dot6_from_pad_cell
```

See [Table 10-4](#) for a description of these values.

- -pin_order_file *file_name*

An optional value pair that is used to reference an existing [pin_order_file : file_name](#). When this option is used, the specified pin_order_file will be validated during the [check_design_rules](#) command instead of being created. If you create this file from a different tool or you re-order the one that was created from a previous run of Tessent Shell, you must point to your file using this option to have it used. You can avoid having to manually reorder this file by reading in a DEF file using the [read_def](#) command.

- -pad_io_ports *pad_io_ports*

An optional value pair used to specify the ports on the current design that are to be equipped with boundary scan cells. This option is only used when the design level is physical_block

or sub_block as specified using the [set_design_level](#) command. The pad_io_ports string is a Tcl list of names of one or more ports, or a collection containing one or more ports of the current design.

Examples

The following example use the -cell_options option to set the sample_only option on a set of ports. Notice that if you have power and ground ports in your design and the design level is top, as specified by the [set_design_level](#) command, you must set the function attribute on them to power and ground as shown here.

```
> set_boundary_scan_port_options [get_ports HIO*] \
    -cell_options sample_only
> set_attribute_value [get_ports vdd*] -name function -value power
> set_attribute_value [get_ports vss*] -name function -value ground
```

Related Topics

[get_boundary_scan_port_option](#)

[report_boundary_scan_port_options](#)

[set_bus_handling](#)

Context: dft -edt, patterns -scan

Mode: analysis

Specifies the bus contention results that you desire for the identified buses.

Usage

```
set_bus_handling {Pass | Fail | Abort} {-All | -Pass | -Fail | -Bidi | -ABort | [-Instances  
instance_name...] | bus_gate_id#... | net_pathname...}
```

Description

Specifies the bus contention results that you desire for the identified buses.

The set_bus_handling command preassigns the contention check handling result that you desire during simulation for the buses that you specify. Upon invocation, the tool automatically calculates the bus contention handling as documented under the [set_contention_check](#) command description. The tool rejects (from the internal test pattern set) ATPG-generated patterns that can cause bus contention. The set_bus_handling command lets you override the automatic contention calculations, thereby changing whether or not the tool performs a simulator-based check using such a pattern.

The tool resets the bus contention handling back to the automatically calculated value whenever you make a change to the modeling that requires the tool to perform a complete re-analysis of the contention mutual exclusivity (such as changing the net resolution or the pin constraints).

Caution

 Overriding the automatically calculated contention check handling results can cause trouble downstream if there is a problem with the design that required modification due to the bus contention.

Arguments

- **Pass**

A literal that specifies for the tool to not perform bus contention evaluations on the buses that you identify and to treat them as if they had passed. This allows the tool to retain patterns that it would otherwise reject due to a contention check failure.

- **Fail**

A literal that specifies for the tool to treat the buses that you identify as if they had failed the bus contention evaluations. This causes the tool to reject patterns that it would otherwise retain due to passing the contention check.

- **Abort**

A literal that specifies the bus aborted the bus contention evaluations before determining whether the bus passed or failed. This can be used with the `analyze_bus -Drc_check` command to verify ATPG constraints which you have added to correct bus failures.

- **-All**

A switch that specifies for the tool to change all buses to a specified state reset.

- **-Pass**

A switch that specifies for the tool to change to specified state for the buses that have previously passed the [E10](#) DRC rule.

- **-Fail**

A switch that specifies for the tool to change to specified state for the buses that have previously failed the [E10](#) DRC rule.

- **-Bidi**

A switch that specifies for the tool to change to specified state for bidi buses identified by the [E10](#) DRC rule.

- **-ABort**

A switch that specifies for the tool to change to specified state for the buses that have previously aborted the [E10](#) DRC rule.

- **-Instances *instance_name***

An optional switch and repeatable string that specifies the instances where the included buses are to be changed to the specified state.

- ***bus_gate_id#***

A repeatable integer that specifies the gate identification numbers of the buses whose contention handling you want to override. If a bus is cascading, you must specify the dominant bus.

- ***net_pathname***

A repeatable string that specifies a net name of the buses to be changed to a specified state.

Examples

The following example turns the bus contention checking off, allowing the bus to pass the evaluations. However, this action can cause trouble in the future if there is a problem with the design that required modification due to the bus contention.

report_bus_data 321

```
/FA1/ha1/XOR1/OUT/ (321) handling=fail type=strong #Drivers=4
  Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
  BUS Drivers: 156(SW) 252(SW) 307(SW) 308(SW)
```

set_bus_handling pass 321
report_bus_data 321

```
/FA1/Ha1/Xor1/OUT/ (321) handling=pass type=strong #Drivers=4
Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
BUS Drivers: 156(SW) 252SW) 307(SW) 308(SW)
```

Related Topics

[report_bus_data](#)

[set_contention_check](#)

set_bus_simulation

Context: dft -edt, patterns -scan

Mode: analysis

Specifies whether the tool uses global or local bus simulation analysis.

Usage

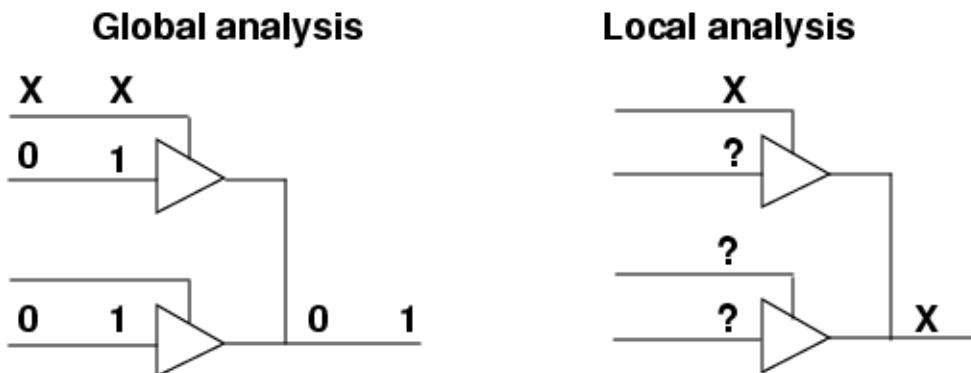
set_bus_simulation [Global | Local]

Description

Specifies whether the tool uses global or local bus simulation analysis.

This command simplifies typical back end verification flows by providing the option to turn off global analysis and use only the values that can be determined by inspecting the immediate input gates (tri-state devices or TSDs) driving a bus. [Figure 6-1](#) shows an example of how the setting of this command effects pattern creation.

Figure 6-1. Global versus Local Bus Simulation Analysis Example



Global analysis assumes a consensus value (0 or 1) is driven onto the bus when one or more of the tri-state enables is X and all inputs to the tri-states are the same (either all 0 or all 1). This most closely matches real hardware behavior. Local analysis assumes an X is driven onto the bus if any of the tri-state enables is X, regardless of the values on the inputs to the tri-states. It is more pessimistic, but more closely matches typical Verilog UDP behavior.

This command is provided to resolve compatibility issues with tools that do not perform global analysis. For example, if your design includes instances of TSDs for gating signals onto a bus, and their behavior in a timing-based simulator is as shown on the right in the preceding figure, using global analysis for pattern creation can result in simulation mismatches when you perform timing-based verification of the test patterns. You can avoid these mismatches by using this command to specify local bus simulation analysis for pattern creation.

Tip The [report_environment](#) command will display the analysis type currently in effect.

Arguments

- Global
Specifies for the tool to perform global bus simulation analysis. This is the default.
- Local
Specifies for the tool to perform local bus simulation analysis.

Related Topics

[report_environment](#)

set_bypass_chains

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.

Usage

```
set_bypass_chains {-Bypass_chain_number chain_number}  
[-Edt_chains {chain_name...} [-Pins bypass_input bypass_output]]
```

Description

Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.

This command allows you to customize the configuration of the bypass chains; for example, to meet unique routing requirements.

If you use this command to customize the bypass chains, you must specify all the EDT bypass chains.

Arguments

- **-Bypass_chain_number *chain_number***

A required switch and positive integer pair that identifies the EDT channel to connect to the specified scan chains to form an EDT bypass chain.

Note

 When you specify the number of channels using the set_edt_options command -Channels switch, the channels are numbered consecutively starting at 1.

- **-Edt_chains {*chain_name...*}**

An optional switch and repeatable string pair that specifies the internal scan chains to connect in series to form the specified bypass chain. The chains are connected from channel input to channel output in the order specified.

- **-Pins *bypass_input bypass_output***

An optional switch and double string that specifies the input and output pins for bypass chains that exist in the design netlist before EDT logic is generated.

Note

 Scan chains and bypass chains must use the same input and output pins.

Examples

Example 1

The following example sets up the EDT logic for a design with eight scan chains named *chain1* through *chain8*. The design is setup with two EDT channels and chains 1, 5 and 6 are connected

in series to channel 1 to form bypass chain 1 and the results display. Any number in the Bypass# column with *no connections* next to it represents an undefined bypass chain.

```
set_edt_options -channels 2
set_bypass_chains -bypass_chain 1 -edt_chains chain1 chain5 chain6
report_bypass_chains

//      Reporting user-defined bypass chains:
//
//      Bypass#  EDT chains
//      -----  -----
//      1        chain1, chain5, chain6
//      2        no connections
//
```

The following commands define the remaining bypass chain and displays the completed user-defined bypass configuration.

```
set_bypass_chains -bypass_chain 2 -edt_chains chain2 chain3 chain4 chain7 chain8
report_bypass_chains

//      Reporting user-defined bypass chains:
//
//      Bypass#  EDT chains
//      -----  -----
//      1        chain1, chain5, chain6
//      2        chain2, chain3, chain4, chain7, chain8
//
```

Example 2

The following example shows a dofile segment that sets up the scan chains and bypass chains for EDT logic and then shows the resulting EDT logic file.

```
// Internal scan chains
add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains chain2 grp1 scan_in2 scan_out2
add_scan_chains chain3 grp1 scan_in3 scan_out3
add_scan_chains chain4 grp1 scan_in4 scan_out4
//
// Bypass scan chains
set_edt_options -channels 2 -bypass use_existing_bypass_chains
set_bypass_chains -bypass_chain 1 -pins scan_in1 scan_out1 -edt_chains \
chain1 chain2
set_bypass_chains -bypass_chain 2 -pins scan_in2 scan_out2 -edt_chains \
chain3 chain4
```

Muxes are added for only the internal scan pins that are also bypass chain pins as shown in the following EDT logic file.

```

module TOP_edt_bypass_logic (edt_bypass, edt_channels_in, edt_channels_out,
                           edt_scan_in, edt_scan_out, edt_bypass_in, edt_bypass_out);
  input      edt_bypass;
  input [1:0] edt_channels_in;
  output [1:0] edt_channels_out;
  output [3:0] edt_scan_in;
  input [3:0] edt_scan_out;
  input [3:0] edt_bypass_in;
  input [1:0] edt_bypass_out;

  assign edt_scan_in[0] = edt_bypass ? edt_channels_in[0] : edt_bypass_in[0];
  assign edt_scan_in[1] = edt_bypass ? edt_channels_in[1] : edt_bypass_in[1];
  assign edt_scan_in[2] = edt_bypass_in[2];
  assign edt_scan_in[3] = edt_bypass_in[3];

  assign edt_channels_out[0] = edt_bypass ? edt_scan_out[0] : edt_bypass_out[0];
  assign edt_channels_out[1] = edt_bypass ? edt_scan_out[1] : edt_bypass_out[1];
endmodule

```

Related Topics

[report_bypass_chains](#)
[reset_bypass_chains](#)
[set_edt_options](#)

set_capture_clock

Context: dft -edt, dft -scan, dft -test_points, patterns -scan,
patterns -scan_diagnosis

Mode: setup, analysis

Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.

Usage

```
set_capture_clock [primary_input_pin] [-Atpg [-Allow_multiple_cycles]] | ANY
```

Description

Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.

This command affects the capture clock that is used during DRC simulation and may be useful for designs with certain types of test logic that must be initialized properly to pass scan chain tracing.

The set_capture_clock command specifies the name of the capture clock the tool uses during random pattern simulation. You can specify the name of either a specific pin or a clock procedure in a test procedure file that identifies the pin. In either case, the pin must be a currently defined clock pin. Also, the capture clock that you specify cannot have a pin constraint.

If you do not specify a capture clock with this command, the tool sets the capture clock to none. If there is no specified capture clock and there is only one clock in the circuit that is not a set or reset line, the tool sets that clock as the capture clock during rules checking and displays a warning message that identifies the capture clock.

When you add a capture clock, the tool checks for pulse-always or pulse-in-capture clocks. If any exist, the tool does not create the capture clock and issues an error. For example:

```
add_clocks 0 CLK1 -pulse_in_capture
add_clocks 0 CLK2
set_capture_clock CLK2 -atpg
// Error: Clock CLK2 cannot be defined as capture clock in the presence
of pulse-in-capture clock CLK1.
```

When you add a pulse-always or pulse-in-capture clock, the tool checks for a user-defined capture clock, a warning message is issued, and the user-defined capture clock is reset to none. For example:

```
add_clocks 0 CLK2
set_capture_clock CLK2 -atpg
add_clocks 0 CLK1 -pulse_always
```

```
// Warning: set_capture_clock is reset as consequence of adding a pulse-
in-capture clock.
```

The `set_clock_options` command's `-pulse_always` and `-pulse_in_capture` switches behave the same way as `add_clocks`. The warning or error will be reported and the capture clock resets when a user-defined capture clock exists.

These two checks only report when you define synchronous pulse-always or pulse-in-capture clocks with a capture clock specified. You can define an asynchronous free running clock in the presence of a capture clock.

You can use the [report_environment](#) command to list the capture clock and the [report_clocks](#) command to identify the current list of clocks.

Note

 This command applies only to random pattern generation unless you include the `-Atpg` switch; then it applies to all scan patterns created during the ATPG process.

The `set_capture_clock` command specifies a particular clock as the capture clock during random pattern generation or, optionally, ATPG. For random pattern generation alone, the command takes effect only if you previously issued a “[simulate_patterns](#) `-source random`” command.

When you have specified a capture clock with this command, the tool may pulse other defined clocks, but it does not use them for capture. You can specify the capture clock by providing the name of a specific pin in a test procedure file that identifies a currently defined clock pin. The capture clock that you specify cannot have a pin constraint.

You can undo a previously specified capture clock choice by issuing this command without any arguments. This will make all clocks available for clocking and capture.

Note

 During ATPG, the capture clock specified with this command is ignored if one or more named capture procedures are available and their use is enabled with “`set_capture_procedure on`” (the default).

Arguments

- *primary_input_pin*

An optional string that specifies the name of the primary input pin you want to assign as the capture clock.

- `-Atpg`

An optional switch that directs the tool to use the specified capture clock as the only capture clock for all patterns created during the ATPG process, not just random patterns. This option also forces the tool to pulse the capture clock only once per capture cycle per pattern, unless you also use the `-allow_multiple_cycles` switch, causing the tool to generate combinational

and clock sequential patterns (no RAM sequential or chain test patterns). If specified in Setup mode, this switch turns on some additional design rule checks (DRCs).

When using this switch in Setup mode, all named capture procedures (NCPs) are disabled by default when entering a non-Setup mode. In addition, the `create_patterns` command can then create only patterns with the specified capture clock. After entering a non-Setup mode, you can manually enable NCPs by using the “`set_capture_procedure On`” command. The tool then checks to ensure that each of the NCPs pulse the capture clock once and only once. The tool does not enable those NCPs that do not satisfy this clock sequence.

Note

 When using Tessent TestKompress (EDT On), you must specify this switch to enable the command to work properly with EDT.

- **-Allow_multiple_cycles**

An optional switch that allows the tool to create patterns with multiple pulses of the capture clock. Using this switch disables the DRC stability check that uses the capture clock, thus allowing ATPG to create patterns with multiple clock pulses of the specified capture clock. This switch is available only when you also use the `-atpg` switch.

- **ANY**(Tessent FastScan and Tessent TestKompress only)

An optional string that specifies any clock can be a capture clock. This is the default if `primary_input_pin` is not specified.

Examples

The following example specifies a capture clock:

```
add_clocks 1 clock1
set_capture_clock clock1
set_system_mode analysis
set_random_patterns 612
```

Related Topics

[add_clocks](#)
[delete_clocks](#)
[report_clocks](#)
[simulate_patterns](#)
[report_environment](#)
[set_capture_procedures](#)
[read_patterns](#)

set_capture_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies how the tool globally handles the data capture of state elements that have C3 and C4 rule violations.

Usage

```
set_capture_handling {-Ls {Old | New | X} -Te {Old | New | X}}
```

Description

Tip

 This command should rarely be used. For most designs, other than those that require mixed capture handling, you should not use this command but rather use split capture handling. Split capture handling is automatically set if needed when you use the [create_patterns](#) command. Alternatively, you can directly set split capture handling using the [set_split_capture_cycle](#) command.

Specifies how the tool globally handles the data capture of state elements that have C3 and C4 rule violations.

The set_capture_handling command gives you some ability to globally change how the tool simulates data capture in the presence of C3 and C4 clock rules violations. C3 and C4 clock rules checks ensure that a clock line cannot capture data affected by the clock, and that any data the clock does capture does not affect the clock line itself. The tool does not normally allow C3 or C4 data capturing violations during simulation.

For information on the C3 and C4 rules, refer to the “[Clock Rules \(C Rules\)](#)” section.

You can use this command to set the global data capture behavior for level sensitive (-Ls) or trailing edge (-Te) devices. For each device type, you can specify for the tool to simulate old, new, or X data at the source points. You can override the global data capture handling for individual state elements by using the [add_capture_handling](#) command.

Arguments

- **-Ls {Old | New | X}**

A switch and literal pair that specifies how you want the tool to handle the data capturing of level-sensitive state elements. The literal choices are as follows:

Old — A literal that specifies for the tool to determine the output value of a level-sensitive source state element by using the data that existed prior to the current clock cycle. The tool then passes this value to the state element’s sink state elements. This is the default behavior of the tool upon invocation.

New — A literal that specifies for the tool to determine the output value of a level sensitive source state element by using the data from the current clock cycle. The tool then passes this value to the state element's sink state elements. The tool limits the scope of this capture handling effect to the circuitry between the source and sink points. The tool will not propagate the newly-captured effect past the sink point.

X — A literal that specifies for the tool to determine the output value of a level-sensitive source state element by using the data from the current clock cycle unless the previous values are different from the current values. If the values differ, the tool passes an unknown (X) value to the state element's sink state elements.

- **-Te {Old | New | X}**

A switch and literal pair that specifies how you want the tool to handle the data capturing of trailing edge sensitive state elements. The literal choices are as follows:

Old — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data that existed prior to the current clock cycle. The tool then passes this value to the state element's sink state elements. This is the default behavior of the tool upon invocation.

New — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data from the current clock cycle. The tool then passes this value to the state element's sink state elements. The tool limits the scope of this capture handling effect to the circuitry between the source and sink points. The tool will not propagate the newly-captured effect past the sink point.

X — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data from the current clock cycle unless the previous values are different from the current values. If the values differ, the tool passes an unknown (X) value to the state element's sink state elements.

Examples

The following example changes the data capture handling for all trailing edge state elements that have C3 and C4 rule violations:

```
set_capture_handling -te new
```

Related Topics

[add_capture_handling](#)
[delete_capture_handling](#)
[report_capture_handling](#)

set_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Enables or disables the tool’s use of a specified subset of named capture procedures when generating test patterns.

Usage

```
set_capture_procedures {ON | OFF} {-All | -Created_capture_procedure |  
-Loaded_capture_procedure | [procedure_name...]}
```

Description

Enables or disables the tool’s use of a specified subset of named capture procedures when generating test patterns.

By default, the tool uses all named capture procedures defined in the test procedure file, except any that fail to pass DRC. The tool never uses capture procedures that fail DRC.

This command is useful when you want to use a subset of the defined capture procedures for certain types of faults, because the remaining capture procedures may be unable to detect the faults.

Note

 The set_capture_procedures command applies to test generation only, not to fault simulation. If you generate test patterns using “set_capture_procedure off”, for example, fault simulation will still simulate those test patterns even if you issue a “set_capture_procedure on” command prior to fault simulation.

Arguments

- ON

A literal that enables use of the specified named capture procedures during test generation. This is the default. If one or more named capture procedures is enabled with this option, the tool does not use the default capture procedure; it uses the named capture procedure(s) exclusively. In this case, if the conditions to test a fault require a particular sequence of clocks or cycles that is not defined in one of the named capture procedures, the fault will be untestable.

Note

 As shown in the examples, use of the “on” option to specify a subset of the defined capture procedures must be preceded by “set_capture_procedure off -all”.

- **OFF**

A literal that disables use of the specified named capture procedures during test generation. If you use this option with “-all” to disable all named capture procedures, ATPG uses the default capture procedure, if present. Otherwise, it runs as if there are no capture procedures defined in the test procedure file.

Note

Use of the “on” option to specify a subset of the defined capture procedures must be preceded by “set_capture_procedure off -all”.

- **-All**

A switch that enables or disables use of all capture procedures defined in the test procedure file. This is the default.

- **-Created_capture_procedure**

A switch that enables or disables use of only the named capture procedures internally created from the [create_capture_procedures](#) command.

- **-Loaded_capture_procedure**

A switch that enables or disables use of only the named capture procedures loaded from the [read_procfile](#) command or the procedure file associated with the scan group definition.

- **procedure_name...**

A repeatable string that enables or disables use of only the named capture procedures with the user-specified name.

Examples

Example 1

The following example loads a fault list, disables use of all capture procedures, enables just the capture procedure named c1_capture, then creates patterns:

```
read_faults clock1_fault_list
set_capture_procedures off -all
set_capture_procedures on c1_capture
create_patterns
```

Example 2

The following example creates named capture procedures from generated patterns, then disables use of loaded capture procedures:

```
create_patterns
create_capture_procedures -pattern internal
set_capture_procedures off -loaded_capture_procedure
```

Related Topics

[create_capture_procedures](#)

[delete_capture_procedures](#)

[report_capture_procedures](#)

[set_drc_handling](#)

set_cell_library_options

Context: unspecified, all contexts

Mode: setup

Sets the specified options for library parsing and design elaboration.

Usage

```
set_cell_library_options [-prefer_cell_library {on | off}] [-force] ]
    [-report_duplicate_models {off | on}]
    [-report_floating_nets {off | on}]
    [-report_errors_only {off | on}]
    [-augment_dft_cell_selections {single_selection_only | on | off}]
    [-default_dft_cell_selection_name default_dft_cell_sel_name]
    [-create_default_dft_cell_selection {on | off}]
    [-liberty_ignore_dont_use {on | off}]
```

Description

Sets the specified options for library parsing and design elaboration.

When you issue the command without arguments, the command reports the current library options settings.

Arguments

- **-prefer_cell_library {on | off}**

An optional switch and literal pair that controls the tool's behavior when reading in a design that has both a Verilog module and a cell library model with the same name. By default the tool chooses the cell library model.

If your design has been elaborated, you cannot change the **-prefer_cell_library** option without losing setup data. If you want to change this option after elaboration, you must use the **-force** switch to explicitly indicate that it is alright for the tool to discard previously defined setup information.

When you specify “off” before issuing the [set_current_design](#) command, the tool chooses the Verilog module.

- **-force (available only in contexts that operate in no_rtl mode)**

An optional switch that specifies to suppress the error message that normally prevents the accidental deletion of previously defined setup information. If you execute the “**set_cell_library_options -prefer_cell_library option**” command after your design has been elaborated, the tool will normally display an error message indicating that if it executes the command, the setup information you have already specified will be discarded. You use this switch to suppress this error message and indicate that it is acceptable for the tool to delete the previously defined setup data.

- **-report_duplicate_models {off|on}**

An optional switch that specifies to issue a separate note for each duplicate model parsed by `read_cell_library`. Otherwise a summary of the total number of duplicate models is issued by default.

- **-report_floating_nets {off|on}**

An optional switch that specifies to issue a separate note for each floating net in each model parsed by `read_cell_library`. Otherwise a summary of the total number of models with floating nets and the total number of floating nets is issued by default.

- **-report_errors_only {off|on}**

An optional switch and literal pair that specifies to suppress warnings and notes and report error messages only. The default is off.

- **-augment_dft_cell_selections {single_selection_only|on|off}**

An optional switch and literal pair that specifies whether to augment parsed `dft_cell_selection` information by adding any legal function not specified within that `dft_cell_selection`.

For example, if a `dft_cell_selection` has nand cell but no xor cell, but a parsed model performs the xor function, then the model will be added as an xor cell in the `dft_cell_selection` and can be used for test insertion and execution of the `get_dft_cell` command.

By default, augmentation occurs only if a single `dft_cell_selection` is parsed in all `read_cell_library` files. When augmentation occurs augmented entries will not be written out by the `write_cell_library` command, and the resulting `dft_cell_selections` will be exactly as parsed. For more information, see the [write_cell_library](#) command.

- **-default_dft_cell_selection_name *default_dft_cell_sel_name***

An optional switch and string that sets the default `dft_cell_selection` to use for test insertion and the execution of the `get_dft_cell` command. Without this command, when two or more `dft_cell_selections` are parsed, test insertion uses the last parsed `dft_cell_selection`, and `get_dft_cell` will error unless a `dft_cell_selection_name` is specified with the `get_dft_cell` command. This option sets the default `dft_cell_selection` to use for both test insertion and the execution of the `get_dft_cell` command.

The `default_cell_sel_name` must be the same as one of the names from the “`dft_cell_selection (dft_cell_selection_name) ()`” statements parsed when the `read_cell_library` command is executed. If `default_cell_sel_name` is “” then the behavior reverts to the behavior before any execution of the “`set_cell_library_options -default_dft_cell_selection_name`” command.

- **-create_default_dft_cell_selection {on|off}**

An optional switch and literal pair that determines whether a default `dft_cell_selection` is created when no `dft_cell_selection` is parsed.

- **-liberty_ignore_dont_use { on | off }**

An optional switch and literal pair that determines whether “cell_type = prohibited;” is placed inside the corresponding Tesson model representing a Liberty cell containing “dont_use” inside the Liberty cell group for that cell. In Liberty, “dont_use” indicates that a synthesis tool should not insert that cell into a netlist when optimizing, and the default of -liberty_ignore_dont_use is “off”, which causes “cell_type = prohibited;” to be placed inside the model so that Tesson test tools will not insert such a cell to create test logic needed for testing. To prevent the inclusion of “cell_type = prohibited” inside the corresponding (same name) Tesson model for a Liberty cell containing “dont_use”, you must issue this command and set this switch to off before the read_liberty command containing the cells, to have any impact on the resultant Tesson cell library models.

Examples

In the following example, no Liberty Cell inside liberty_source_file, or any Liberty Cell of a subsequent “read_liberty” command, will have “cell_type = prohibited;” placed inside the corresponding Tesson read_cell_library model due to a “dont_use” in the Liberty syntax.

```
set_cell_library_options -liberty_ignore_dont_use on
read_liberty liberty_source_file
read_cell_library corresponding_tesson_models_filename
write_cell_library final_tesson_cell_library_with_liberty_info_merged_filename
```

In Liberty source given to read_liberty :

```
cell("AN2") {
    dont_use : true ;
```

Tesson write_cell_library for Liberty cell default model output :

```
model AN2 .
...
    cell_type = prohibited;
    simulation_function = and;
...
```

Tesson write_cell_library for Liberty cell after specifying
“set_cell_library_options -liberty_ignore_dont_use on”:

```
model AN2 .
...
    cell_type = and;
    simulation_function = and;
...
```

Related Topics

[read_verilog](#)
[delete_design](#)
[set_design_include_directories](#)

[set_current_design](#)

[set_design_macros](#)

[read_cell_library](#)

[delete_cell_library](#)

[set_cell_model_mapping](#)

Context: dft -scan

Mode: setup, analysis

Prerequisites: You can only override the mapping for scan models of the same scan type. For example, a mux-DFF scan model can only be replaced by another mux-DFF scan model.

Overrides the non-scan to scan model mapping defined by the tool.

Usage

```
set_cell_model_mapping -New_model scan_model_name
    [-MODULE module_name... | -Instance {pathname... | instance_expression...} |
     -File filename] [-MODEL model_name...] [-Output scan_output_pin_name]
```

Description

Overrides the non-scan to scan model mapping defined by the tool.

You can change the scan model for an individual instance, all instances under a hierarchical instance, all instances in all occurrences of a module in the design, or all occurrences of the model in the entire design. Additionally, you can change the scan output pin of the scan model in the same manner.

Note

 Use this command for changing the existing non-scan to scan model mapping, as defined in the library, and *not* to define the mapping.

Refer to “[Scan Cell and Scan Output Mapping](#)” in the *Tessent Scan and ATPG User’s Manual* for conceptual information on this topic.

Arguments

• -New_model *scan_model_name*

A required switch and string that specify a new library model for the specified design instances.

• -MODULE *module_name*... | -Instance {*pathname*... | *instance_expression*...} | -File *filename*]

An optional switch and string or repeatable string that specify the design instances whose models should be mapped to the library model specified by the -New_model switch. If this switch is omitted, all instances of the model are specified with the -Model option.

-Module — Assigns the library model specified by -New_model to either all compatible design instances within the module *module_name* or to just the design instances of the model specified with the -Model option.

-Instance — Specifies the pathname(s) of either a hierarchical or library instance(s).

For a hierarchical instance, assigns the library model specified by *-New_model* to either all compatible design instances within this hierarchical instance or to just the design instances of the model specified with the *-Model model_name* option.

For a library instance, maps either all compatible instances to the specified *-New_model* library model or to just the instances of the model specified with the *-Model model_name* option. The repeatable string can be in the form of an absolute instance pathname or a pathname in regular expression form.

-File — Specifies the name of the file containing pathname(s) of instances that should be mapped to the *-New_model* model, if compatible.

- **-MODEL *model_name***

An optional switch and string that specify the name of an existing model whose design instances are to be mapped to the model specified by *-New_model*. The specified model can be either a non-scan or a scan model. However, the following assumptions are made:

Non-scan models — The new scan model is appropriate for the non-scan model.

Scan model — The old and new scan models are appropriate for at least one common non-scan model.

- **-Output *scan_output_pin_name***

An optional switch and string pair that specify the name of the scan output pin to use in place of the tool-defined scan output pin. The pin/port must have been declared as a scan-out port in the *scan_definition* section of *-New_model*.

set_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies binary sequences for chain test.

Usage

Available when EDT is Off

```
set_chain_test [-SUPpress_capture_cycle {OFF|ON}]  
[{-SEQUENCE {bit_string | OFF} ... [-APPend | -REplace]}]
```

Available when EDT is On

```
set_chain_test [-SUPpress_capture_cycle {OFF|ON}]  
[{-SEQUENCE {bit_string | OFF} ... [-APPend | -REplace]}]  
[-Type {All | Nomask [with_1hot] | Mask [XOr_only | 1Hot_only]}]  
[-NOMASK_First {ON | OFF}]
```

Description

Specifies binary sequences for chain test.

Use this command to add one or more custom sequences to the current chain test or replace the current chain test sequences with all new sequences.

Arguments

- **-SUPpress_capture_cycle {OFF|ON}**

An optional switch and literal pair that suppresses the capture cycle from the chain test patterns. In the capture cycles, constrained pins are forced to their constrained values and cell constraints are justified.

By default, chain test patterns contain a capture cycle when there is no C1 violation. In case DRC detects C1 violations on the level-sensitive ports or set/reset ports, the tool automatically suppresses the capture cycle for the chain test patterns in order to avoid those C1-violated cells to be disturbed and prevent simulation mismatches. You can manually change the default setting (`set_chain_test -Suppress_capture_cycle Off`) to force the chain test patterns to include the capture cycle. Under this situation, the tool masks out the scan cells with a C1 violation on either synchronous port or set/reset ports to prevent mismatches because the C1-violated cells on a synchronous or set/reset port may cause the loading value to be disturbed in the capture cycle. Note that this masking effect could impact the diagnosability of the chain test failure but should not impact the overall test quality.

This switch also suppresses the capture cycle of the first pattern (`edt_setup`) before chain and scan test patterns when the low-power EDT decompressor or input channel pipelining is in use. For more information, see “[Low-Power Test](#)” in the *Tessent TestKompress User’s Manual*.

- **-SEQUENCE** *bit_string* | OFF

An optional, repeatable literal and string pair that specifies a binary sequence to shift into the scan chains, starting with the most significant bit.

When EDT is OFF, if the length of a specified bit string sequence is shorter than a scan chain, the specified sequence is repeated to fill the rest of the scan chain. If the specified sequence is longer than the scan chain, the extra less significant bits are truncated.

When EDT is ON, the tool attempts to get the sequence to occur once in one-hot patterns. This is not a repeating sequence.

Specifying OFF disables custom sequences. This allows you to revert back to using the default chain test.

- **-APPEND** (Available when EDT is Off)

An optional switch that specifies to add the specified sequences to the tool's current chain test sequences.

- **-REPLACE** (Available when EDT is Off)

An optional switch that specifies to replace the tool's current chain test sequences with the specified bit sequences. This is the default.

- **-TYPE** All | Nomask [with_1hot] | Mask [XOr_only | 1Hot_only] (Available when EDT is On)

An optional switch and literal that specifies the type of EDT chain test patterns that should be saved to minimize effort during simulation run time. The chain test patterns are saved when the [write_patterns](#) command is issued. The possible literal values for **-Type** are:

All — Saves chain test patterns with all types of masking. This option includes all non-masking and masking patterns as well as patterns to test all unused decoder values when all chain patterns are selected. As a result, the total number of patterns returned by this option may be greater than the sum of all non-masking and masking patterns. For more information on unused patterns, see “[EDT Logic and Chain Testing](#)” in the *Tessent TestKompress User’s Manual*.

Nomask — Only saves the non-masking chain test patterns.

with_1hot — Saves the 1-hot chain patterns along with the no-masked chain patterns.

Caution

 You must *not* use the Nomask option when saving your *production* test patterns. If you save the production pattern set using the “set_chain_test -type nomask” option, it is *very* difficult to identify which chain failed and automatic chain diagnosis is not possible.

If you are *not* using the low-power decompressor and you are constrained by the number of patterns you can simulate, you can run one non-masking pattern to achieve minimum coverage, as shown here:

```
set_chain_test -type nomask  
write_patterns pattern_filename -pattern_sets chain -serial -end 0
```

If you *are* using the low-power decompressor, it is safest to run all non-masking patterns so that every chain is tested, as shown here:

```
set_chain_test -type nomask  
write_patterns pattern_filename -pattern_sets chain -serial
```

Mask — Only saves the masking chain pattern.

XOr_only — For the Xpress compactor, saves only flexible masking chain patterns.

Note: This does not apply to the basic compactor.

1Hot_only — Saves only 1-hot masking chain patterns and two nomask patterns.

If you do not save 1-hot patterns (*Nomask* or *Mask [XOr_only]* options), Tesson Diagnosis may not have enough information to run chain diagnosis successfully. Without 1-hot patterns, chains in a compactor are scanned out together and the tool cannot identify the specific chain that contains the error. If you do not want to save 1-hot patterns but still want to run chain diagnosis, you must communicate to Tesson Diagnosis the failing chains using the following commands:

```
read_patterns scan_patterns  
diagnose_failures <failure_file> -faulty_chain <faulty_chain_name>\<faulty_chain_type>
```

This approach assumes internal failing chain information was reported through another method. For more information, see the [read_patterns](#) and [diagnose_failures](#) commands.

For more information on masking patterns, see “[Understanding Scan Chain Masking in the Compactor](#)” in the *Tesson TeskKompress User’s Manual*.

- **-NOMASK_First {ON | OFF}** (Available when EDT is On)

An optional switch that specifies whether to place EDT non-masking chain patterns first. The default is OFF which means that 1-Hot masking chain patterns are ordered first. The -Nomask_First ON option is only valid with -Type All—the option is ignored if used with -Type Mask.

Examples

Example 1

Assume a design contains two scan chains: chain1 is ten scan cells long and chain2 is eight cells long.

The following example saves the tool’s standard chain test patterns to the file my_chain_test.ascii. The sequences (minus X-padding) that you would see in the file are shown after the example commands.

```
set_system_mode analysis
write_patterns my_chain_test.ascii -pattern_sets chain -replace
```

Pattern 0

chain1: 0011001100

chain2: 00110011

Example 2

The following example replaces the current chain test sequences with three new sequences and saves the resultant chain test patterns:

```
set_system_mode analysis
set_chain_test -sequence 01 -sequence 1
set_chain_test -sequence 1001 -append
write_patterns my_chain_test.ascii -pattern_sets chain -replace
```

Now, in the my_chain_test.ascii file, you would see the following three chain test sequences (minus X-padding):

Pattern 0

chain1: 0101010101

chain2: 01010101

pattern 1

chain1: 1111111111

chain2: 11111111

pattern 2

chain1: 1001100110

chain2: 10011001

set_checkpointing_options

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables checkpointing and specifies the associated options.

Usage

```
set_checkpointing_options [OFF | ON {[-PATTERN_FIle filename]
[-PATTERN_FOrmat {Ascii | Binary}] [-Faultlist_file filename] [-Replace]
[-Period period]})]
```

Description

Enables checkpointing and specifies the associated options.

Checkpointing is when the tool automatically saves test patterns at regular periods, referred to as checkpoints, throughout the pattern creation process. This is useful when ATPG takes a long time and there is a possibility it could be interrupted accidentally.

The set_checkpointing_options command specifies the filename and time period in which the tool writes test patterns during test pattern generation.

You can use the -Faultlist_file option to save the fault list. In the case of an interruption, if you have saved the generated patterns, you can use this fault list to avoid the resimulation step. When you specify the -Faultlist_file option, the tool saves the checkpointed fault list, including the fault class and sub-class of each fault, to the specified file. Aborted faults are also written to the file.

You can use the set_checkpointing_options command multiple times within a session and each time modify only the options you want to change. The tool retains your unmodified settings from the previous commands.

Note

 Although you can issue the set_checkpointing_options command in any mode, the tool only writes out a checkpoint file in analysis mode with the set_checkpointing_options command set to On.

For additional information on checkpointing, refer to “[Checkpointing Setup](#)” in the Tessent Scan and ATPG User’s Manual.

Arguments

- OFF

A literal that specifies for the tool not to use the checkpointing functionality during test pattern generation. Patterns are not written to any file. This is the invocation default.

- **ON**
A literal that specifies for the tool to use the checkpointing functionality. The tool writes test patterns that it generates to the file that you specified with the `set_checkpointing_options` command.
- **-PATTERN_FILE *filename***
An optional switch and string pair that specifies the name of the file into which you want to write the test patterns during test pattern generation.
- **-PATTERN_FORMAT {Ascii | Binary}**
An optional switch and literal pair that allows the pattern files to be saved in binary or ASCII format. Binary is the default format.
- **-Faultlist_file *filename***
An optional switch and string pair that allows the fault list to be saved.
- **-Replace**
An optional switch that forces the tool to overwrite the file if it already exists. If you use the `-Replace` option and the tool does not create any new test patterns, the tool does not update the file.
- **-Period *period***
An optional switch and integer pair that specifies the number of minutes between each write of the test patterns. The default is 100 minutes.

Examples

The following example stores the generated test patterns every 120 minutes in a file called `my_checkpoint_file`. After each 120-minute interval, the tool creates a new file until the ATPG process ends.

```
set_system_mode analysis
set_checkpointing_options on -pattern_file my_checkpoint_file -period 120
create_patterns
```

Related Topics

[write_flat_model](#)
[write_patterns](#)
[create_patterns](#)
[write_faults](#)

[set_clock_controller_pins](#)

Context: dft -logic_bist

Mode: setup

Specifies the connection information for the clock controller pins.

Usage

```
set_clock_controller_pins
  {shift_clock | shift_clock_en | shift_capture_clock | scan_en |
  | capture_en | lbist_en |
  capture_procedure_index | diag_clock_en }
  {internal_pin_name... | -no_connection} [-append]
```

Description

Specifies the connection information for the clock controller pins.

This command is used with the Hybrid TK/LBIST flow. Refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

If you issue this command multiple times with the same pin type, then the last specified pin name(s) are used. To specify multiple pin names corresponding to a pin type, specify all the names in the same command, or use the -append switch to add to the list of already specified pins.

The pins you specify with this command are connected to the single top-level LogicBIST controller. This command is global in context and not specific to the current EDT block.

Manual specification of the clock controller pins is not required when Tessent OCC core instances have been added. In this case, only the capture_procedure_index connection is required.

Alternatively, you can specify the -no_connection switch instead of internal_pin_name in an RTL IP-creation flow (skeleton flow), where the clock controller is not yet available in the dummy design.

Note

 The internal_pin_name string and **-no_connection** switch are mutually exclusive.

Arguments

- **shift_clock internal_pin_name**

An optional literal and repeatable string that specifies the clock controller shift clock input. This pin is to be driven by the LogicBIST shift clock output from the controller in LogicBIST mode.

- **shift_clock_en *internal_pin_name***

An optional literal and repeatable string that specifies the clock controller shift clock enable input. This pin is to be driven by the LogicBIST shift clock enable output from the controller in LogicBIST mode.

- **shift_capture_clock *internal_pin_name***

An optional literal and repeatable string that specifies an input on the clock controller that needs to receive a clock that pulses during shift and capture, unlike the shift_clock that only pulses during the shift mode operation. This pin is driven by the LogicBIST shift capture clock output from the controller in LogicBIST mode. If this pin is not specified, then the output port will not be created. This option is mutually exclusive with shift_clock.

Specifying shift_capture_clock will clear any previously specified shift_clock connections.

- **scan_en *internal_pin_name***

An optional literal and repeatable string that specifies the clock controller scan enable input. This pin is to be driven by inverted LogicBIST capture enable output from the controller in LogicBIST mode.

- **capture_en *internal_pin_name***

An optional literal and string that specifies the capture enable trigger LBIST clock enable input. This is typically connected to LBIST capture enable output.

- **lbist_en *internal_pin_name***

An optional literal that specifies the LogicBIST enable pins.

- **capture_procedure_index *internal_pin_name***

An optional literal that specifies the connections to the signal that identifies the NCP used for the current pattern. Then internal_pin_name string identifies the pins in the clock controller for the NCP index specified from MSB to LSB.

A capture_procedure_index pin type has to be specified once per clock controller to add the list of NCP index pins for that clock controller. It is not possible to group the connections for the capture_procedure_index pin type for all clock controllers in a single command. The -append switch must be used to add to the pin specification for prior controllers.

- **diag_clock_en *internal_pin_name***

An optional literal and repeatable string that specifies the clock controller diagnosis clock enable input. The LogicBIST controller diag_clock_en output drives this pin. If you are using the shift_clock_en clock controller pin, you need a diag_clock_en connection to perform LogicBIST diagnosis. The diag_clock_en signal is only allowed when using “set_edt_options -single_bypass on.” When using shift_clock, the tool injects TCK during diagnosis through the shift_clock pin; if you want to inject TCK, you should use this pin.

- **-no_connection**

An optional switch used instead of internal_pin_name.

When the `-no_connection` switch is used, the design clock controller pin names need not be specified and no connections will be made. The pins are available at the LogicBIST IP boundary. This switch can be used in an RTL IP-creation flow, where the clock controller is not yet available in the dummy design.

- `-append`

An optional switch used to append to a previous list of pin names, instead of replacing. Without the `-append` switch, repeat usage of the `set_clock_controller_pins` command overwrites the previous list of pins for the specified pin type.

Examples

Example 1

The following example specifies using `controller/scan_en` as the scan enable input pin on the clock-controller interface:

```
set_clock_controller_pins scan_en controller/scan_en
```

Example 2

In this example, assume that the clock controller is not available in the design. The following example specifies the clock controller's shift clock but does not make any connection to the clock-controller interface:

```
set_clock_controller_pins shift_clock -no_connection
```

Example 3

This example connects the `lbist_en` output of the LogicBIST controller to the LogicBIST enable pin on multiple clock controllers. Note that the arguments should be specified in a list surrounded by braces.

```
set_clock_controller_pins lbist_en \
    {cc_clk1/lbist_mode cc_clk2/lbist_mode cc_clk3/lbist_mode}
```

Example 4

This example shows how to connect the NCP index pins of multiple clock controllers using the `-append` switch.

Note

 You must specify a connection for each clock controller separately as the `capture_procedure_index` pin type requires a list of pin names be declared for each clock controller.

```
set_clock_controller_pins capture_procedure_index \
    [get_pins cc_clk1/ncp*]
set_clock_controller_pins capture_procedure_index \
    [get_pins cc_clk2/ncp*] -append
set_clock_controller_pins capture_procedure_index \
    [get_pins cc_clk3/ncp*] -append
```

Issuing a report_clock_controller_pins command will show the following:

```
//  command: report_clock_controller_pins
//
//  PinType          Connection
//  -----
//  LBIST enable    cc_clk1/lbist_mode cc_clk2/lbist_mode
//                  cc_clk3/lbist_mode
//  Capture procedure index  cc_clk1/ncp_index[1] cc_clk1/ncp_index[0]
//                          cc_clk2/ncp_index[1] cc_clk2/ncp_index[0]
//                          cc_clk3/ncp_index[1] cc_clk3/ncp_index[0]
//
```

Related Topics

[report_clock_controller_pins](#)

[set_lbist_controller_options](#)

[set_clock_controls](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables and disables the clock control definitions in the test procedure file.

Usage

```
set_clock_controls {ON | OFF}  
[-ENFORCE_off_conditions_outside_capture_if_pulse_in_capture_source {ON | OFF}]
```

Description

Enables and disables the clock control definitions in the test procedure file.

When enabled, the clock definitions are used for ATPG. Clock control definitions cannot be used simultaneously with Named Capture Procedures (NCPs).

For more information, see “[Support for Internal Clock Control](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **ON**

Required literal that enables clock control definitions. This is the default setting unless Named Capture Procedures (NCPs) are enabled in the test procedure file. When NCPs are enabled, the internal clocks are automatically disabled.

- **OFF**

Required literal that disables clock control definitions. Unless the internal clocks are controlled by NCPs, turning clock control off leads the tool to treat the internal clocks as directly controllable and ignore the constraints and condition bits imposed on ATPG by the clock control definitions. Any generated patterns are invalid since the clock controller requirements modeled using the clock control definitions are completely ignored by ATPG and simulation. The ability to turn off clock controls is provided only for debug purposes, and is typically used to determine if a fault is untestable because of the clock control constraints (and if the fault is testable if those constraints are not present).

- **-ENFORCE_off_conditions_outside_capture_if_pulse_in_capture_source {ON | OFF}**

An optional switch and keyword pair that enables turning off the following default behavior: when the source clock of a clock control is an pulse-in-capture clock, the tool uses condition bits to ensure that the internal clock is off in cycles beyond those generated by ATPG even if the source clock is pulsed beyond that point. This prevents disturbing the captured values.

When generating patterns in the presence of internal clocks controlled by using clock control definitions, the tool must either decide whether to use condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern or assume that disabling the source clock will suppress further internal clock pulses. For

example, consider that the clock control definition for a given internal clock specifies three condition bits: B1 to control the clock pulsing in the first cycle, B2 to control it pulsing in the second cycle, and B3 to control it pulsing in the third cycle. If ATPG generates a one-cycle pattern that pulses this clock, the tool encodes B1=1. The tool then has the option of encoding B2=0 and B3=0, or leaving them unspecified to be random-filled.

If the clock control definition has no source clock defined or has an asynchronous free-running clock, then the tool must encode those two additional condition bits to ensure that any pulses the OCC receives beyond the first pulse are suppressed and the internal clock only pulses once. On the other hand, if the clock control definition has a source clock that can be controlled by the tester and is now defined as pulse-in-capture or pulse-always, this source clock will be pulsed only once, and there is no need to waste encoding capacity to encode 00 into those two bits because the internal clock pulses only once regardless of the values loaded into B2 and B3.

The tool decides whether to use condition bits to suppress further internal clock pulses as follows:

If a clock control definition has no source clock, or its source clock(s) are asynchronous free-running, the tool applies the condition bits needed to suppress additional internal clock pulses beyond the sequential depth of the generated pattern. You have no way to change this.

If generating retargetable patterns (“set_current_mode -type internal”), the tool applies the condition bits to suppress additional internal clock pulses. You cannot change this because the clocks at the core boundary typically do receive additional clock pulses when retargeted and merged with patterns for other cores.

If the source clock(s) defined in a clock control block are pulse-in-capture, the tool applies the condition bits to suppress additional internal clock pulses. You can override this behavior by using “set_clock_controls on -enforce_off_conditions_outside_capture_if_pulse_in_capture_source off.” This reduces the number of specified bits and may improve EDT compression. However, if you use this option, you must be sure that the source clock will not be pulsed additional times than ATPG has pulsed it. Do not use the “set_external_capture_options -fixed_cycles” or “set_external_capture_options -minimum_cycles” commands when writing your patterns, since those options pulse the pulse-in-capture clocks when adding the padding cycles. Note that capture cycle padding is used for the type-3 LPCT controller, and as such, you should not use “set_clock_control on -enforce_off_conditions_outside_capture_if_pulse_in_capture_source off” in that case either.

If none of the previous conditions are satisfied and the source clock(s) are defined and completely under ATPG control with no constraints, then ATPG does not apply the condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern.

Examples

The following example is a debugging flow that disables all clock control definitions in the test procedure file and generates test patterns:

```
set_clock_controls off  
create_patterns
```

Related Topics

[report_clock_controls](#)

set_clock_gating

Context: dft -scan, dft -test_points

Mode: setup

Specifies clock gating cells whose unconnected test-enable ports need to be connected to either the scan enable signal or a specified signal (pin).

Usage

```
set_clock_gating {ON | OFF} [object_spec]
    [-Port_to_connect port_name [-INVert]]
    [-clock_port clock_port_name]
    [-clock_out_port clock_out_port_name]
```

Description

Specifies clock gating cells whose unconnected test-enable ports need to be connected to either the scan enable signal or a specified signal (pin).

Note

 When wrapper analysis is used, and the clock gater drives wrapper cells, then the port will be connected to a composite signal that consists of scan_enable or'ed with the appropriate mode enable (for example, ext_mode_enable). Alternatively, when multiple scan enables are used, the appropriate scan enable signal will be selected based on the type of scan cells each clock gater is driving.

The tool considers an unconnected port to be a port that is not connected to any net or is tied low (TIE0), tied high (TIE1), tied unknown (TIEX), or tied high impedance (TIEZ).

The tool does one of the following:

- Automatically checks if the test-enable port of each recognized clock gating instance or module is unconnected and, if unconnected, connects it to the scan enable signal.
- Automatically checks if the specified port of each specified instance is unconnected and, if unconnected, connects it to the scan enable signal.

Note

 This command is cumulative. Every “set_clock_gating on” command invocation adds to the list of clock gaters that will be connected, and every “set_clock_gating off” command invocation removes clock gaters from the list.

Arguments

- **ON**
A literal that specifies to connect the unconnected test-enable port of the provided instances or modules. If no *object_spec* is provided, it applies to all identified clock gaters. This is the default.
- **OFF**
A literal that specifies to not allow connecting the unconnected test-enable port of the provided instances or modules. If no *object_spec* is provided, it applies to all identified clock gaters.
- ***object_spec***
A Tcl list of one or more instances or modules or a collection of instances or modules.
- **-Port_to_connect *port_name*[-INVert]**
An optional switch and string pair that specifies the unconnected port of the clock gating cell to be connected to either the scan enable signal or to the specified signal.
-INVert — By default, the expected input value for the unconnected port is set to 1 during the shift cycle. This switch specifies that the expected input value is set to 0 during the shift cycle.

The -Port_to_connect option is required when you issue the command with the on switch and provide *object_spec*. For the command to succeed, the port names specified with this option must exist on all the modules/instances defined with *object_spec*. To add modules that have different names for their test-enable or clock pins, issue the command multiple times.

The default tool behavior is equivalent to “set_clock_gating on”, meaning that all identified clock gaters will be connected. The clock gaters are initially identified either through auto-identification or are inferred from the cell library.

The tool marks the unconnected clock gaters and identifies their test-enable pins according to the following priority:

- First, each clock gater explicitly provided with the “set_clock_gating on” command will be marked, and the port specified by the -port_to_connect switch will be identified as its test_enable pin.
- Next, every instance whose module’s cell_type equals clock_gating_and or clock_gating_or will be marked, and the port with the test_enable or test_enable_inv attribute will be identified as test-enable. If the clock gater is not driven by any synchronous clock it will not be marked.
- Finally, all auto-identified clock gater instances will be marked with one of their unconnected inputs identified as test-enable.

- **-clock_port *clock_port_name***

An optional switch and string pair that specifies the clock port of the clock gating cell. Use this option when the clock gating device is to be connected to an enable signal associated with the clock domain.

- **-clock_out_port *clock_out_port_name***

An optional switch and string pair that specifies the clock out port of the clock gating cell. Use this option when the clock gating device has multiple outputs.

Examples

The following example demonstrates the use of the set_clock_gating command:

```
set_clock_gating on /inst1/cgc1 -port_to_connect te  
set_clock_gating off /inst1/cgc2
```

The first set_clock_gating command connects the /inst1/cgc1/te pin to the relevant scan enable signal, marks all clock gaters whose cell module has cell_type clock_gating_and or clock_gating_or and identifies test-enable signal based on test_enable or test_enable_inv pin attribute. The second set_clock_gating command specifies that the /inst1/cgc2 instance will not be connected. All remaining auto-identified clock gaters will be marked and one of their unconnected enable pins will get connected to the appropriate scan enable.

Related Topics

[report_clock_gating](#)

set_clock_gating_enable

Context: dft -scan, dft -test_points

Mode: setup

Associates clocks that control clock gating devices with enable signal drivers.

Usage

```
set_clock_gating_enable enable_signal_driver_pathname [primary_input]
    [-Active {High | Low}]
    [-Clock_domain clock_signal_driver_pathname...]
```

Description

Associates clocks that control clock gating devices with enable signal drivers.

This command assumes that the clocks controlling the clock gating devices (whose enable ports are to be automatically connected) were added to the design with the [add_clocks](#) command.

The tool connects the unconnected enable ports of clock gating devices to the enable signal associated with the controlling clocks of these clock gating devices. The clock gating devices, the enable ports of the clock gating devices, and their clock ports must be specified with the [set_clock_gating](#) command.

Arguments

- ***enable_signal_driver_pathname***

A required string, single-element Tcl list or a collection that specifies either a new top-level port, an existing top-level enable port or an existing internal output pin to be connected to the unconnected enable port of a clock gating device.

The *enable_signal_driver_pathname* argument can be either a top-level enable port or an internal instance pin. If an internal instance pin is specified, the tool attempts to locate its primary input driver by tracing back to a primary input via a simple path; that is, through only inverters or buffers. If automatic identification of the primary input driver fails, the value of the *primary_input* option is used, if provided. Identification of the primary input is needed to generate the ATPG dofile and test procedure files.

- ***primary_input***

An optional Tcl list or a collection that specifies a top-level enable port for an internal instance pin provided by the *enable_signal_driver_pathname* argument. The specified top-level port is used when generating the ATPG dofile and test procedure files.

- **-Active {High | Low}**

An optional switch and literal pair that specify whether the enable signal is active low or active high. The default setting is high.

- **-Clock_domain** *clock_signal_driver_pathname...*

An optional switch and a repeatable string that specifies the defined clock(s) controlling clock gating devices whose unconnected enable ports are to be connected to the specified enable signal's driver.

Examples

The following example specifies the default clock gating enable signal to be used when the tool does not find an enable signal associated with the specified clock domain.

```
set_clock_gating_enable x  
set_clock_gating_enable y -clock_domain a
```

In this example, the signal x will be used to enable all clock gaters except for those in clock domain a, which will use signal y.

set_clock_off_simulation

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables or disables the simulation where all clock primary inputs are at their “off” value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.

Usage

`set_clock_off_simulation ON | OFF`

Description

Enables or disables the simulation where all clock primary inputs are at their “off” value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.

When simulating this event, the capture data for inputs is provided to leading edge triggered flip-flops.

Tip If you use ATPG Expert (create_patterns), you do not need to issue this command for better handling of DRC violations. ATPG Expert automatically enables it when needed. You may get better results by letting ATPG Expert decide when to use this command except in the case of transparent latch handling.

Note that the set_clock_off_simulation command affects ATPG (subsequent `create_patterns` commands) but not logic simulation (subsequent `simulate_patterns` commands).

Arguments

- **ON**
A literal that performs ATPG for all clock primary inputs at their “off” values.
- **OFF**
A literal that performs ATPG for all clock primary inputs at their active values. This is the default.

Examples

The following example enables clock_off simulation prior to creating patterns:

```
set_system_mode analysis
set_clock_off_simulation on
create_patterns
```

Related Topics

[set_split_capture_cycle](#)

[set_clock_options](#)

Context: dft, patterns

Mode: setup

Allows you to specify multiple options for a single clock.

Usage

```
set_clock_options object_spec [-label]  
  [[-period period {s | ms | us | ns | ps | fs}]  
   | [-reference ref_obj_spec [-reference_inv ref_inv_obj_spec]  
   | [-freq_multiplier freq_multiplier] [-freq_divider freq_divider]]]  
   [-pulse_always {ON | OFF} | -pulse_in_capture {ON | OFF}] [-off_state {0 | 1}]  
   [-dft_inject_node dft_inject_node] [-no_dft_connections]
```

Description

Allows you to specify multiple options for a single clock.

The `set_clock_options` command allows you to edit options for an existing clock without having to delete the clock and create a new clock with the [add_clocks](#) command.

Arguments

- ***object_spec***
A required string that specifies an existing clock using the name of a port, pin, pseudo_port, label, or a collection of one port, or one pin, one pseudo_port, or one label object. The *object_spec* cannot be a branch.
- **-label *label***
An optional switch and string pair that specifies an arbitrary alias name for the clock.
- **-period *period***
An optional switch and real number pair that specifies the base clock period. Setting the clock period changes the clock type from synchronous source to asynchronous source. Setting the clock period to a null string changes the clock type from asynchronous source to synchronous source.
- **[s | ms | us | ns | ps | fs]**
An optional literal that specifies a time unit if you use the -period switch. If you don't use this switch, the default is "ns."
- **-reference *ref_obj_spec* ("patterns -ijtag" only)**
An optional switch and string pair that specifies the source primary input port that is connected to the internal clock pin you specified with the *object_spec* argument. You use this switch only for an existing source clock on a pseudo_port, not for a port. Setting the *ref_obj_spec* to a null string changes the clock type from generated to synchronous source.

- **-reference_inv *ref_inv_obj_spec* (“patterns -ijtag” only)**

An optional switch and string pair that specifies a second primary input port if the **-reference** pin is a differential pin. You can use this switch only if you also use the **-reference** switch or if the *object_spec* is already a generated clock.

- **-freq_multiplier *freq_multiplier* (“patterns -ijtag” only)**

An optional switch and string pair that defines the multiply factor of the clock frequency between the reference pin and the internal clock pin specified with the *object_spec* argument. You can use this switch only if you also use the **-reference** switch or if the *object_spec* is already a generated clock.

- **-freq_divider *freq_divider* (“patterns -ijtag” only)**

An optional switch and string pair that defines the division factor of the clock frequency between the reference pin and the internal clock pin specified with the *object_spec* argument. You can use this switch only if you also use the **-reference** switch or if the *object_spec* is already a generated clock.

- **-pulse_always {ON | OFF}**

An optional switch and literal pair that specifies the *object_spec* as a pulse-always clock. A pulse-always clock is pulsed in every cycle. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting. You can use this switch only when *object_spec* is an existing source clock. This switch has no default value.

- **-pulse_in_capture {ON | OFF}**

An optional switch and literal pair that specifies the *object_spec* as a pulse-in-capture clock. A pulse-in-capture clock is always pulsed during the capture window. You can use this switch only when *object_spec* is an existing source clock. This switch has no default value.

- **-off_state {0 | 1}**

An optional switch and literal pair that specifies the off state of the clock. You cannot use this switch if *object_spec* is an existing async source or if you use the **-period** switch. This switch has no default value.

- **-dft_inject_node *dft_inject_node* (“dft” only)**

An optional switch and string pair that specifies where the DFT object affecting the clock (for example, a TCK injection multiplier) should be inserted along the clock path. You can introspect the status of this switch with the “[get_clock_option -dft_inject_node](#)” command and switch.

- **-no_dft_connections**

A Boolean switch that instructs the tool to not infer DFT connections to this clock. When choosing the clock for a dedicated wrapper cell or a test_point in the [insert_test_logic](#) command, the clock used by the majority of the flip-flops in the fanin or fanout of the node is normally used. If that clock was added with the **-no_dft_connections** switch, it is not chosen and instead the next majority clock is used. If all clocks the fanin or fanout were

added with the `-no_dft_connections` switch, the largest clock domains in the design without this switch are selected.

Examples

The following example creates a synchronous clock with a label, then uses the `set_clock_options` command to change the clock to asynchronous and assign a different label name:

```
SETUP> add_clocks 0 I3 -pulse_always -label sync3
```

```
SETUP> report_clocks
```

```
User-defined Clocks (1) :  
Sync and Async Clocks  
=====
```

Name	Label	Off State	Constraints	Internal
'I3'	sync3	0	Pulse always	No

```
SETUP> set_clock_options I3 -label async3 -period 100
```

```
SETUP> report_clocks
```

```
User-defined Clocks (1) :  
Sync and Async Clocks  
=====
```

Name	Label	Off State	Constraints	Internal	Period
'I3'	async3	-	Asynchronous	No	100.00ns

Related Topics

[add_clocks](#)

[delete_clocks](#)

[get_clock_option](#)

[get_clocks](#)

[report_clocks](#)

set_clock_restriction

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether ATPG can create patterns with more than one active capture clock.

Usage

```
set_clock_restriction {Clock_po | ON | OFF | Domain_clock [-Edge_interaction  
| -Any_interaction] | [-PULSE_interacting_clocks {OFF | ON}]  
[-MAX_SKew {Half_cycle |  
One_cycle | Multiple_cycles}]  
[-SAME_clocks_between_loads {OFF | ON}]  
[-COMPAtible_clocks_between_loads {OFF | ON}]  
[-IGNORE_unobservable_cells {ON | OFF}]
```

Description

Specifies whether ATPG can create patterns with more than one active capture clock.

The set_clock_restriction command changes the default behavior of ATPG regarding the creation of test patterns that have more than one active clock line.

Note

 Pulse-always clocks are excluded from clock restriction checking when you choose Clock_po or ON. For example, if you choose ON, and two pulse-always clocks are defined, then only one clock is allowed in addition to the pulse-always clocks.

If you choose Domain_clock, the tool forces pulse-always clocks to be compatible with other clocks.

Arguments

- **Clock_po**

A literal that specifies that multiple clocks are allowed to be active for clock_po patterns. For non-clock_po patterns, the option is equivalent to the ON option which only allows a single active clock per capture cycle.

For more information on clock_po patterns, see the “[Clock PO Patterns](#)” section in the *Tessent Scan and ATPG User’s Manual*.

- **ON**

A literal that specifies for the ATPG to create only patterns with, at most, a single active clock.

The ATPG treats equivalent clocks as a single clock, however, it treats multiple active clocks as a conflict condition and performs an exhaustive search for conditions necessary to detect the fault with no more than one active clock. Faults which the ATPG detects at

primary outputs (POs) that connect to clocks must also satisfy this condition. Faults detected at clock POs that require multiple active clocks for detection, require that you use the Clock_po option. You can accomplish this by setting the clock restriction to Clock_po and then re-running the ATPG.

During the bus contention prevention analysis portion of the ATPG, Tessent FastScan and Tessent TestKompress turn off any clock pins that the ATPG does not require for fault detection.

- **Off**

A literal that specifies for the ATPG process to create patterns with as many clocks on as it requires to detect faults. Using this option may cause race conditions due to multiple active clocks. Prevent these race conditions by specifying the On argument.

For Tessent FastScan and Tessent TestKompress — These tools concurrently apply any clocks they require for a given pattern. They display a message at the end of the ATPG run to indicate the number of patterns that had more than one active clock. When you change the clock restriction to off, these tools reset the ATPG untestable faults to undetected-uncontrolled.

- **Domain_clock**

A literal that specifies for ATPG to create patterns with compatible clocks on. This enables the tool to generate patterns that pulse compatible scan chain capture or system clocks at the same time. Normally, only one scan chain capture or system clock is pulsed in a cycle. The tool treats two clocks as compatible if their domains are completely independent of each other; that is, they can be in their “on” state at the same time without causing clock skew problems or race conditions.

The tool automatically analyzes circuit data from DRC to determine the compatible clocks. Therefore, if you use this option prior to DRC, the tool defers execution until DRC.

Note

 The tool’s analysis of clock domains only considers Tie-0 or Tie-1 gates that are Tie-0 or Tie-1 in both combinational and sequential ATPG.

- **-Edge_interaction**

An optional switch that, when the Domain_clock setting is in effect, specifies for the tool to determine if two clocks are compatible (C) or incompatible (I) based on their edge interactions, as summarized in [Table 6-4](#). The table covers the case where a source memory element is clocked by one clock, a sink memory element is clocked by a different clock, and the source and sink interact (the sink can capture data from the source).

Note

 Take care when interpreting what the “C” and “I” mean: “C” means only that the tool treats the result of the particular combination as being unaffected by clock skew. “I” means the tool treats the result of the particular combination as susceptible to skew.

You can issue the `set_clock_restriction` command with only this switch, and the tool will retain whatever clock restriction setting (`Clock_po`, `On`, `Off` or `Domain_clock`) is currently in effect. This switch has no effect on the analysis used for ATPG unless `Domain_clock` is in effect; however, it does affect the output of the [report_clock_domains](#) command.

This is the default upon invocation of these tools.

Table 6-4. Default Clock Compatibility Summary at a Clock Domain Boundary

		Sink			
		LE¹	TE²	AH(TE)³	AL(LE)⁴
S	LE⁵	I	C	C	I
O	TE⁶	C	I	I	C
U	AH(LE)⁷	I	C	C	I
R	AL(TE)⁸	C	I	I	C

1. LE = Edge sensitive element captures on PI clock's leading edge
2. TE = Edge sensitive element captures on PI clock's trailing edge
3. AH(TE) = Active high, level sensitive element captures on PI clock's trailing edge
4. AL(LE) = Active low, level sensitive element captures on PI clock's leading edge
5. LE = Edge sensitive element changes on PI clock's leading edge
6. TE = Edge sensitive element changes on PI clock's trailing edge
7. AH(LE) = Active high, level sensitive element changes on PI clock's leading edge
8. AL(TE) = Active low, level sensitive element changes on PI clock's trailing edge

ATPG Expert automatically sets the appropriate switch based upon the fault type. For stuck-at faults, ATPG Expert always sets “`set_clock_restriction domain_clock-edge_interaction`”.

- `-Any_interaction`

An optional switch that, when the `Domain_clock` setting is in effect, specifies for the tool to do a more conservative analysis of the clocking relationship between interacting source and sink memory elements in different clock domains. If the source is clocked by one clock, the sink is clocked by a different clock, and the two elements interact (the sink can capture data from the source), this switch causes the tool to treat the two clocks as incompatible. In other words, all the conditions shown in [Table 6-4](#) result in a determination of incompatibility.

You can issue the `set_clock_restriction` command with only this switch, and the tool will retain whatever clock restriction setting (`Clock_po`, `On`, `Off` or `Domain_clock`) is currently

in effect. This switch has no effect on the analysis used for ATPG unless Domain_clock is in effect; however, it does affect the output of the [report_clock_domains](#) command.

ATPG Expert automatically sets the appropriate switch based upon the fault type. For transition faults, ATPG Expert always sets “set_clock_restriction domain_clock -any_interaction -compatible_clocks_between_loads on”.

- **-PULSE_interacting_clocks {OFF|ON}**

An optional switch and literal pair that specifies whether the interacting clocks can be pulsed simultaneously during test. By default, the tool disables the pulsing of interacting clocks. The ON option allows pulsing pairs of interacting clocks during test generation. Note that this switch is effective only when you also specify the Domain_clock option.

When this switch is ON, the tool masks some state elements located at the inter-clock domain in order to avoid simulation mismatches. Therefore, you will notice more X values at the state elements than when this switch is OFF. The behavior of cross-domain path masking depends on the setting of the max clock skew, as described below for the -Max_skew switch.

Note

The tools treats user-defined compatible clocks (created with the [add_synchronous_clock_group](#) command) as compatible clock domains and does not mask out cross-clock domain paths between compatible clocks.

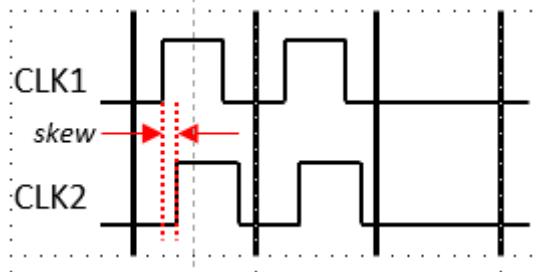
- **-MAX_SKW {Half_cycle | One_cycle | Multiple_cycles}**

An optional switch and literal pair that specifies the maximum skew between interacting clocks pulsed during capture. The maximum skew determines whether the observation state element should be masked or if fault simulation can give credit for detection. This option is effective only when the -Pulse_interacting_clocks option is enabled.

The maximum skew settings impact fault simulation as follows:

- Half_cycle — Only masks cross-clock domain paths of the same edge. [Figure 6-2](#) shows half-cycle skew:

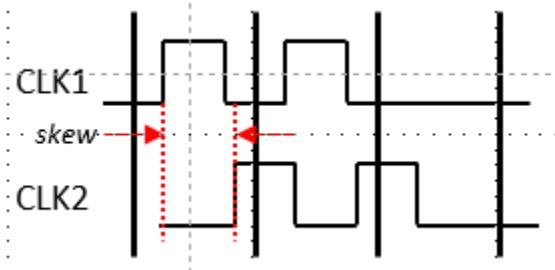
Figure 6-2. Half-cycle Skew



- One_cycle — Masks cross-clock paths of the same clock edge, leading edge to trailing edge of the same cycle, and trailing edge to the leading edge of the next cycle. It allows the transition to be tested by launching at the leading (trailing) edge of one clock and capturing at the next cycle leading (trailing) edge of another clock.

Figure 6-3 shows one-cycle skew:

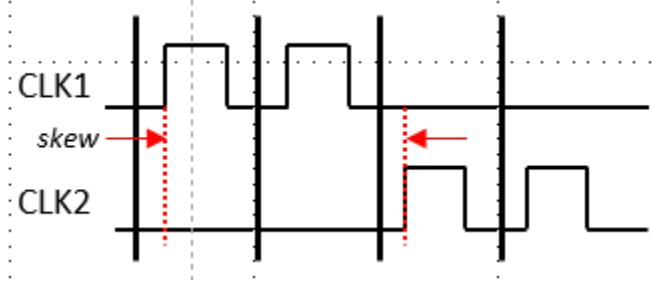
Figure 6-3. One-cycle Skew



- `Multiple_cycles` — Masks all cross-clock domain paths. Tool assumes multiple clock cycles between all cross-clock domain paths and simulates them as asynchronous.

Figure 6-4 shows multiple-cycle skew:

Figure 6-4. Multiple-cycle Skew



The default maximum skew settings are as follows:

- Static fault models (stuck-at and bridging) — Default setting is `Half_cycle`, although the default is `One_cycle` when you also use the `-Any_interaction` switch.
- At-speed fault models (transition and path delay) — When the `-same_clocks_between_loads` switch is disabled, interacting clocks may be pulsed in adjacent cycles. In this case, the default setting is `One_cycle`, which allows transition from one edge to the same edge in the next cycle. When the `-same_clocks_between_loads` switch is enabled, interacting clocks are pulsed in every cycle and all clock interactions should be considered invalid. To reduce the chance of simulation mismatches, the default setting is `Multiple_cycle`.

You can use the `-max_skew` switch to overwrite the default clock skew to either prevent simulation mismatch (by specifying more clock skew than the default) or obtain better test coverage (by specifying less clock skew than the default). However, when changing the max skew to be less than the default, it must match with the design clock timing; otherwise, there is the possibility of simulation mismatch.

- **-SAME_clocks_between_loads {OFF|ON}**

An optional switch and literal pair that specifies whether to use the same clocks in every cycle between load/unload operations (during capture cycles). This switch is useful for at-speed test generation when you want each pattern to test for delay faults in only one clock domain. When clocks are pulsed between loads, they will all be pulsed for every cycle between loads.

Generally, each test pattern created with this switch enabled will pulse the *same* clocks in every test cycle that pulses clocks. Multi-load patterns are an exception: for a multi-load pattern, the switch applies to each load/unload interval independently. Therefore, the clocks used in a particular interval might not be the clocks used in another interval in the same multi-load pattern.

When enabled, this switch has no effect on pattern simulation. However, the tool will transcript a warning message if a pattern violates the restriction. You can issue the `set_clock_restriction` command with only this argument, and the tool will retain whatever other clock restriction setting (`Clock_po`, `On`, `Off` or `Domain_clock`) is currently in effect.

OFF — A literal that disables the requirement to use the same clocks in every cycle within a load/unload interval during pattern creation. This is the invocation default.

ON — A literal that specifies to use the same clocks in every cycle within each load/unload interval during pattern creation.

- **-COMPAtible_clocks_between_loads {OFF|ON}**

An optional switch and literal pair that specifies that only compatible clocks are allowed to pulse between load/unload operations (during capture cycles). This switch is used with the `domain_clock` option.

OFF — A literal that turns off the compatible clock requirement. When set, incompatible clocks are not allowed to pulse in the same cycle, but they are allowed to pulse at different capture cycles.

ON — A literal that allows only compatible clocks to pulse within capture cycles. When set, incompatible clocks are never allowed to pulse between load/unload operations.

The “`-compatible_clocks_between_loads on`” switch and literal pair is not compatible with the `-edge_interaction` switch or `-pulse_interacting_clocks`. When you specify “`-compatible_clocks_between_loads on`” and `-edge_interaction`, the tool issues an error message. It also issues an error message when you specify “`-compatible_clocks_between_loads on`” and `-pulse_interacting_clocks`.

Tip

If you want to create patterns that test only intra-clock domain faults, and you want to allow multiple non-interacting clock domains to be tested simultaneously in order to reduce pattern count, issue the following command prior to creating patterns:

```
set_clock_restriction domain_clock -any_interaction  
-compatible_clocks_between_loads on
```

- **-IGNORE_unobservable_cells {ON | OFF}**

An optional switch and literal pair that bypasses checking any unobservable nonscan or scan cells for clock interaction purposes. You can issue the `set_clock_restriction` command with only this argument, and the tool will retain whatever other clock restriction setting (`Clock_po`, `On`, `Off`, or `Domain_clock`) is currently in effect.

ON — A literal that enables bypassing the check for unobservable nonscan or scan cells.

OFF — A literal that disables bypassing the check for unobservable nonscan or scan cells.

For example, if two clocks interact on a state element that is unobservable due to pin constraint or cell constraint, then the tool does not consider these two clocks incompatible for this statement when this switch is set to ON.

Examples

The following example checks the current clock restriction setting, adds the additional restriction that the tool use the same clocks in every cycle within each load/unload interval during pattern creation, then changes the basic setting from `clock_po` to “on”:

report_environment

```
...
clock restriction =      clock_po
...
```

set_clock_restriction -same_clocks_between_loads on
report_environment

```
...
clock restriction =      clock_po, same_clocks_between_loads
...
```

set_clock_restriction on
report_environment

```
clock restriction =      ON, same_clocks_between_loads
```

Related Topics

[add_synchronous_clock_group](#)

[report_clock_domains](#)

set_compactor_connections

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies the scan chains for the tool to connect to a particular scan channel output.

Usage

```
set_compactor_connections {-CHANnel channel_number} {-CHAIns {chain_name...}}
```

Description

Specifies the scan chains for the tool to connect to a particular scan channel output.

The set_compactor_connections command specifies a list of scan chains whose outputs you want the tool to compact into the specified EDT scan channel output. The tool makes these connections using one or more stages of XOR gates in the compactor circuitry. If you use this command to specify the compactor connections for one scan channel and group of scan chains, you must use it for all the channels and chains in the design.

When generating the EDT IP, the tool uses the sequence of [add_scan_chains](#) commands to connect the EDT hardware to the scan chains. You can change the order of the add_scan_chains commands in your dofile to change how they are connected to the decompressor and compactor. However, this method changes both the decompressor and compactor connections for a particular chain.

Alternatively, you can use the set_compactor_connections command to override the tool's default connections and explicitly define the connections between scan chains and compactor. Using this command allows you to change the compactor connections without changing the default decompressor connections to those chains.

If you have dual configurations, you can still define the compactor connections using the set_compactor_connections command, but only for the configuration that uses all scan channels. The set_compactor_connections command is not tied to a specific configuration since you only need to define connections once for each channel.

Arguments

- **-CHANnel *channel_number***

A required switch and positive integer pair that identify the scan channel to connect to the specified group of scan chains.

When you specify the number of channels using the set_edt_options command *-Channels* switch, the tool numbers the channels consecutively starting at 1.

- **-CHAIns chain_name...**

A required switch and repeatable string pair that specify the internal scan chains whose outputs the tool should connect through the compactor circuitry to the specified scan channel output.

Examples

Example 1

For a design with eight scan chains named “chain1” through “chain8”, the following example sets up the EDT logic to have two scan channels. It then connects the outputs of chain1, chain5 and chain6 through the compactor to channel 1 and displays these connections. Any channel listed with “no connections” next to it still needs to be defined. The example goes on to define the remaining compactor connections and displays the completed configuration.

```
set_edt_options -channels 2
set_compactor_connections -channel 1 -chains chain1 chain5 chain6
report_compactor_connections

// Reporting user-defined compactor connections:
//
// Channel    Chains
// -----
//      1      chain1, chain5, chain6
//      2      no connections
//

set_compactor_connections -channel 2 -chains chain2 chain3 chain4 chain7 chain8
report_compactor_connections

// Reporting user-defined compactor connections:
//
// Channel    Chains
// -----
//      1      chain1, chain5, chain6
//      2      chain2, chain3, chain4, chain7, chain8
//
```

Example 2

In the following example, compactor connections are defined for a dual configuration. Note, the connections are only defined for the configuration that uses all scan channels.

```
set_current_edt_configuration config_high
set_edt_options -input 1 -output 1
set_current_edt_configuration config_low
set_edt_options -input 3 -output 3
set_compactor_connections -channel 1 -chains ...
set_compactor_connections -channel 2 -chains ...
set_compactor_connections -channel 3 -chains ...
```

Related Topics

[report_compactor_connections](#)
[reset_compactor_connections](#)

[**set_edt_options**](#)

set_config_value

Context: unspecified, all contexts

Mode: all modes

Sets the value of an element in the configuration data.

Usage

```
set_config_value config_spec
  [-in_wrapper wrapper_object_spec]
  [-id {position | id_name | all} | -value_id {position | value_name | all}]
  {values | -specify_default | -unspecify | -comment comment}
  [-partition partition]
```

Description

Sets the value of an element in the configuration data.

See the [get_attribute_value_list](#) command for strategies for introspection of object types.

Arguments

- ***config_spec***

A required value that specifies the name of a configuration element or a collection containing one configuration element. When ***config_spec*** is a collection, the **-in_wrapper** option cannot be used because the object contains all of the information about the configuration element. In order to use the **-in_wrapper** option, the ***config_spec*** value must be a name and the name must be relative to the specified wrapper element.

You can use a positional *id* for any level of hierarchy. You can also replace the *id* of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)”, and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- **-in_wrapper *wrapper_object_spec***

An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. The *wrapper_object_spec* value is the name of a wrapper or a collection containing a wrapper element. The name in ***config_spec*** is relative to the specified parent wrapper.

- **-id *position* | *id_name* | *all***

An optional switch and value pair that specifies that the data to set is the *id* of a wrapper or of a repeatable property. When using **-id all**, the command expects a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable **Id** declared for it. In this case, the value is supplied as a Tcl string which is the case for most wrappers and repeatable properties you will set.

When you have configuration elements with more than one *id*, you can set one *id* at a time using either its position or, if it has metadata, the *id* of the **Id** wrapper that declares the *id*. A

position is represented with an integer surrounded by <>. For example, <0> is the first id, <1> is the second, and so on.

- **-value_id** *position | value_name | all*

An optional switch and value pair that specifies that the data to set is the value of a property, repeatable property, or is a row inside a [DataWrapper](#) or a [CsvWrapper](#). If the -id or -value_id options are not specified, “-value_id all” is the default.

When using “-value_id all”, the tool expects a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the value is expected to be a Tcl string which is the case for most properties you will set.

When you have configuration elements with more than one value, you can set one value at a time using either its position or, if it has metadata, the *id* of the [Value](#) wrapper that declares the value. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on.

- **values**

An string or Tcl list of strings that defines the value at which to set the [Ids](#) or [Values](#) of properties or wrappers. The tool also accepts collections where a value is expected and automatically extracts the name of the object, or list of object names.

You use a list only when using “-id all” or “-value_id all” unless the configuration element has metadata and there is only one non-repeatable id or value declared for it; otherwise, you use a string.

- **-specify_default**

An Boolean option that specifies that the property is to be set to its default value. This option is only allowed on properties.

- **-unspecify**

An Boolean option that specifies that the property is to be unspecified and thus return to its default value. This option is only allowed on properties.

- **-comment** *comment*

An switch and value pair that specifies the string comment for a configuration element.

- **-partition_name** *partition_name*

An optional Boolean option that specifies that the data requested is the name of the partition in which the configuration element exists. The example shows the returned value based on which partition the configuration element exists in.

An option and value pair that specifies the parent wrapper in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications ([DftSpecification](#) and [DefaultsSpecification/DftSpecification](#)) are located. You can access the Core wrappers comprising the Tesson Core Description syntax in the tcd partition. Using the partition names meta: and meta:tcd allows you to access the

metadata configuration elements that define the default and tcd partitions.

[Table 6-5](#) summarizes the available partition names.

Table 6-5. Available Partition for Introspection

Partition name	Configuration data contained
(unspecified)	When the -partition option is unspecified, the Default partition is accessed which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tesson Core Description syntax is accessed
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification / DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

Examples

Example 1

The following example uses a foreach loop to set the properties inside a wrapper

```
> set int [get_config_el Interface \
- in /DftSpecification(corea,rtl)/IjtagNetwork/HostScanInterface(ijtag) ]
> foreach {prop val} {scan_in mySI scan_out mySO tck myTCK} {
    set_config_val $prop -in $int $val
}
> report_config_data $int
Interface {
    tck : myTCK;
    scan_in : mySI;
    scan_out : mySO;
}
```

Example 2

This example uses `set_config_value` to set a list of cell name in the [DftSpecification/force_creation_of_rtl_cells](#) property:

```
> set spec [create_dft_spec]
> set_config_value force_creation_of_rtl_cells -in $spec {and2 or2 inv}
> report_config_data $spec -l 1
DftSpecification(ctrl_core,rtl) {
    IjtagNetwork {
        // Not shown
    }
    force_creation_of_rtl_cells : And2, Or2, Inv;
    MemoryBist {
        // Not shown
    }
}
> get_config_value force_creation_of_rtl_cells -in $spec
And2 Or2 Inv
```

Related Topics

[get_config_value](#)

[get_defaults_value](#)

[report_config_data](#)

set_contention_check

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.

Usage

```
set_contention_check {{ON | OFF} | CAPture_clock} [-Warning | -Error] [-Bus | -Port | -All] [-BIdi_retain | -BIDI_Mask] [-NO_Floating {BIdi_only | All_busses}] [-ATpg | -CATpg] [-NOVerbose | -Verbose | -VVerbose]}
```

Description

Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.

When a bused output of a tri-state driver (or switch) that is driving an X caused by an X on its enable is encountered, the contention is not reported if the data input to the driver is at the same level as other drivers on the bus. Also, the simulated value of the bus gate is resolved to a binary value if it can do so without tracing through additional bus gates.

Arguments

- **ON**

A literal that performs contention checking during simulation. The tool does not propagate captured data effects. This is the default.

If the set_split_capture_cycle command is on and there are multiple capture clocks in the design, the set_contention_check ON command performs a contention check not only before the capture but after the capture as well.

- **OFF**

A literal that specifies to not perform contention checking during simulation.

Tip

 If you create patterns using ATPG Expert (“create_patterns”), the tool automatically determines whether or not the -Atpg switch is necessary and will set it if simulation determines that too many patterns are being rejected due to contention; therefore, you do not need to specify “set_contention_check on -atpg”. This may, in some cases, be faster than setting it initially if a lot of fault coverage can be obtained using ATPG without it. If you are not sure if the switch is needed, it may be better to try running without it.

- **Capture_clock**

A literal that performs contention checking both with and without propagating captured data effects. If a clock, read control, or write control line connects to a bus, the tool also performs bus contention checking with all clocks off prior to the application of the capture clock. The

tool does not consider any contention patterns for fault simulation and does not place any of these patterns into the internal test pattern set.

Tip

i If you create patterns using ATPG Expert (“create_patterns”), the tool automatically determines if the -Catpg switch is necessary and use it only when it is needed; therefore all you need to specify is “set_contention_check capture_clock”. This may, in some cases, be faster than setting it initially if a lot of fault coverage can be obtained using ATPG without it. If you are not sure if the switch is needed, it may be better to try running without it.

- **-Warning**

An optional switch that displays a warning message, but continues simulation if bus contention occurs during simulation. The warning message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- **-Error**

An optional switch that displays an error message and stops the simulation if bus contention occurs.

You can debug contention errors by using the -Error switch to stop simulation at the point of the first contention error.

Using this option, you can then view the simulated values of all gates in the first bus contention pattern by using the report_gates command.

The error message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- **-Bus**

An optional switch that performs contention checking of tri-state driver buses. This is the default.

Tri-state logic allows several bus drivers to time-share a bus. However, if the circuit enables two bus drivers of opposite logic to drive the bus, physical damage can occur. This switch allows the tool to identify these conditions and notify you of their existence.

The tool identifies buses which have circuitry that prevents bus contention and does not check for bus contention problems. This eliminates false bus contention reporting when multiple inputs to a bus are at X. Bus contention that occurs on weak buses does not result in an E4 rules checking violation or pattern rejection during simulation. The tool continues to simulate them as an X state.

- **-Port**

An optional switch that performs contention checking for multiple-port flip-flops and latches. The tool identifies and rejects patterns during which any multiple-port latch or flip-flop has more than one clock, set, or reset input active (or at X).

- **-BIDI_Retain**

An optional switch that rejects patterns during contention checking that cause the direction of bidirectional pins to change following the capture clock.

- **-BIDI_Mask**

An optional switch that causes the fault simulator to modify the input values of bidi pins to avoid bus contention after the capture clock if a device pin changes from input mode to output mode as the clock is applied.

- **-NO_FLOATING {BIDI_only | ALL_busses}**

An optional switch and literal pair that checks for floating buses in addition to contention when bus contention checking is in effect. The BIDI_only option only checks for floating at buses associated with bidirectional pins. The ALL_busses option checks for floating at all buses. This switch enables the E11 DRC with handling set to warning if it was previously set to ignore. (If you have previously set E11 handling to note or error, your setting is retained).

The floating bus checks are always performed during pattern simulation when bus contention checking is in effect. If -Atpg or -Catpg is included, floating buses are avoided during test generation.

The following rules apply:

- Floating checks are performed for strong buses only, not weak buses.
- If a strong bus passes the E11 DRC, the switch does not apply to that bus.
- If a strong bus drives a pull bus, the strong bus is allowed to float if the weak input of the pull bus is set to a known binary value.
- This switch and the -BIDI_mask switch cannot be used together in the same command.
- This switch does not require that a bus have a known binary value to pass the floating check. The bus may be driven to an unknown binary value.
- If a bus is driven by a TIEX gate directly, the TIEX gate is treated as a source that can take on a Z value. As a result, it is impossible to avoid floating this bus and ATPG cannot generate any test patterns.

- **-ALL**

An optional switch that performs contention checking for both tri-state driver buses and multiple-port flip-flops and latches.

- **-ATPG**

An optional switch that forces all buses to a non-contention state, which ensures that test patterns are not created causing bus contention.

After completing normal test pattern generation for a fault, the tool forces all buses to a non-contention state. If the tool cannot satisfy this condition, given the conditions set by the

original pattern, the tool aborts the fault, excludes the pattern from the final test set, and displays a message indicating the number of these aborted faults for each simulation pass. No attempt is made to change the original pattern.

The -Atpg option results in additional effort by the test pattern generator and you should use it only when necessary.

Tip

 If you create patterns using ATPG Expert (“create_patterns”), you may get the best results by not explicitly specifying the -Atpg switch. ATPG Expert will automatically determine if the switch is necessary and utilize it only when it is needed.

- **-CATpg**

An optional switch that uses clock sequential ATPG effort in the capture cycle as well as all other cycles. If neither this switch nor the -Atpg switch is specified, the default will continue to be that there is no ATPG effort for bus contention in any cycle, and ATPG constraints are satisfied deterministically prior to the capture clock event only. Specifying the -Catpg switch without the Capture_clock option is equivalent to using just the -Atpg switch.

Tip

 If you create patterns using ATPG Expert (“create_patterns”), you may get the best results by not explicitly specifying the -Catpg switch. ATPG Expert will automatically determine if the switch is necessary and utilize it only when it is needed.

- **-Verbose**

An optional switch that reports the first reason for each pattern rejected (maximum of 64 messages per parallel pattern invocation).

- **-VVerbose**

An optional switch that reports every reason for each pattern rejected (there is no limit to the number of messages per parallel pattern invocation). The messages will be of the same type issued for the -Verbose switch.

Note

 For large designs, this option may produce thousands of lines of output for each pattern simulated.

- **-NOVerbose**

An optional switch that lets you turn off the -Verbose or -VVerbose effects that may have been set previously.

Examples

Example 1

The following example performs contention checking on multiple-port flip-flops and latches, stops the simulation if any bus contention occurs, and displays an error message. The message indicates the number of patterns rejected and the bus gate on which the bus contention occurred:

```
set_contention_check on -port -error
set_system_mode analysis
create_patterns
```

Example 2

The following example performs contention checking on both multiple-port sequential gates and tri-state buses, stops the simulation if any bus contention occurs, and displays an error message that will indicate the gate on which the contention occurred:

```
set_system_mode analysis
set_contention_check on -all
create_patterns
```

Related Topics

[report_gates](#)
[set_bus_handling](#)
[set_gate_report](#)

set_context

Context: unspecified, all contexts

Mode: all modes

Specifies the current usage context of Tesson Shell.

Usage

```
set_context dft [{-edt [-logic_bist]} | { -scan [-legacy]} | -test_points] [-rtl | -no_rtl]
[-license license_name] [-design_identifier string] [-force]

set_context patterns [-ijtag | -scan| -scan_retargeting | -scan_diagnosis | -failure_mapping]
[-silicon_insight] [-rtl | -no_rtl] [-license license_name]
[-design_identifier string] [-force]
```

Description

Specifies the current usage context of Tesson Shell. You must set the context before you can invoke most other commands in Tesson Shell.

If you attempt to change context and have generated patterns, the tool prompts you to save the patterns with an error message similar to the following:

```
// Error: Current test pattern set has not been saved. Save the
// patterns or use the -force switch.
```

Arguments

- **dft**

A required literal that sets the context to dft. In the basic dft context, you load your design in setup mode and then move to the insertion mode to execute design editing commands such as [create_connections](#), [create_instance](#), and [move_connections](#). When used with the -scan or -edt options, the dft context is used to insert scan chains or insert the EDT controller.

- **patterns**

A required literal that sets the context to patterns which provides functionality related to scan and IJTAG pattern generation, ICL extraction, and scan pattern diagnosis.

- **-scan**

An optional switch that is available in both patterns and dft contexts. This sub-context specifies ATPG and simulation functionality for standard ATPG, Tesson TestKompress, and LogicBIST patterns when specified with the patterns context, and for scan insertion when specified with the dft context.

- **-legacy**

An optional switch that is available in the dft context and must be specified in conjunction with the -scan switch. This sub-context specifies that Tesson Scan is invoked in the legacy mode without hierarchical scan insertion.

- **-edt**
An optional switch that is available in dft context. This sub-context specifies to create and insert EDT IP into the design. When you provide a scan-inserted netlist, the tool analyzes the scan chains and generates and inserts an appropriate EDT controller.
- **-logic_bist**
An optional switch that is available in the dft context and must be specified in conjunction with the -edt switch. This switch enables the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.
- **-test_points**
An optional switch that is available in the dft context and specifies functionality for test point identification and insertion.
- **-ijtag**
An optional switch that is available only in patterns context. This sub-context specifies functionality related to the [IEEE 1687-2014](#) (IJKTAG) standard such as generating IJKTAG patterns by executing PDL commands such that they are retargeted to the current design boundary, and performing ICL extraction. When the [set_current_design](#) command is executed, the presence of an ICL module matching the specified current design name sets the context to ijtag retargeting. If there is no ICL module matching the specified current design name, the ICL extraction mode is enabled. Executing “[set_context](#) -extraction” returns 1 when in the extraction mode.
- **-scan_retargeting**
An optional switch that is available in the patterns context. This sub-context enables scan pattern retargeting functionality for retargeting core-level test patterns at the top level. For more information, see “[Scan Pattern Retargeting](#)” in the *Tessent Scan and ATPG User's Manual*.
- **-failure_mapping**
An optional switch that is available only in the patterns context. This sub-context specifies to reverse map top-level failure files for retargeted scan pattern results, which were generated from within the -scan_retargeting sub-context. Reverse mapping results in a set of core-level failure files that you can diagnose with Tessent Diagnosis. For more information, refer to the *Tessent Diagnosis User's Manual*.
- **-silicon_insight**
An optional switch that is available only in the patterns context. This sub-context enables the SiliconInsight capabilities. For more information, see the *Tessent SiliconInsight User's Manual for Tessent Shell*.
- **-rtl | -no_rtl**
In dft context, these switches specify whether you are performing netlist editing or RTL editing. You must use the -rtl option if you are editing RTL. You can use either option when

editing a netlist but be aware that the [write_design](#) usage model is different between these modes. If you want to write out your edited netlist into a single file, select the `-no_rtl` option.

Note

 By default, the tool enables the preserve escaping in RTL contexts and disables the escaping in non RTL contexts. Switching between `-rtl` and `-no_rtl` contexts in the same tool invocation enables and disables escaping automatically.

In the patterns `-ijtag` context, the `-rtl` switch is inferred if coming from the `dft` context and the `-rtl` option was used when setting the `dft` context. When entering the patterns `-ijtag` context from the unspecified context, `-no_rtl` is assumed. ICL Extraction makes use of quick synthesis to convert any RTL in the fanin or fanout of ICL modules. See the description of the [synthesize_before_analysis](#) and [exclude_from_synthesis](#) Module Built-in Attributes if you are using a `test_setup` procedure for ICL extraction.

In all other contexts, the `-rtl` switch is not allowed and the `-no_rtl` switch is optional because it is always inferred.

In patterns `-ijtag` context with RTL support, during ICL extraction the input cones in front of design ports with matched ICL attributes are synthesized and flattened in order to perform extraction of the paths and end points, which results in an extracted ICL top level description. Checking and debugging of trace issues, IJTAG-related checks and ICL semantic checks are also done at this stage. The same is true for output cones driven by design ports with matched ICL attributes.

- `-license license_name`

An optional switch and string pair that specifies a particular license to use. By default, Tessent Shell acquires the least expensive license available that provides the requested functionality. License names for MemoryBIST and BoundaryScan are accepted without the `-LV`, `-LVNS`, or `-TS` suffix and will default to the LV version of the license.

The `TESSENT_LICENSE_ORDER` environment variable modifies the priority and order of the available licenses. Any available licenses not explicitly listed in the value of the `TESSENT_LICENSE_ORDER` environment variable are appended to the list in their original order.

[Table 6-6](#) lists the supported license names for the available contexts and sub-contexts.

Table 6-6. Contexts Available by License

Context/Sub-Context	License Name(s)
dft -rtl; dft -no_rtl	DesignEditor BoundaryScan-LV BoundaryScan-LVNS BoundaryScan-TS ScanPro IJTAG PLLTest MemoryBIST-LV MemoryBIST-LVNS MemoryBIST-TS SerdessTest TestKompress LogicBIST MissionMode
dft -edt	TestKompress
dft -logic_bist -edt	LogicBIST
dft -scan	Scan ScanPro
dft -test_points	ScanPro LogicBIST

Table 6-6. Contexts Available by License (cont.)

Context/Sub-Context	License Name(s)
patterns -ijtag	IJTAG BoundaryScan-LV BoundaryScan-LVNS BoundaryScan-TS ScanPro PLLTest MemoryBIST-LV MemoryBIST-LVNS MemoryBIST-TS SerdessTest TestKompress LogicBIST MissionMode
patterns -scan	When EDT is used: TestKompress When EDT and LBIST are not used: TestKompress, FastScan When LBIST is used: LogicBIST
patterns -scan_retargeting	TestKompress
patterns -failure_mapping	Diagnosis
patterns -scan_diagnosis	Diagnosis
patterns -siliconinsight	SiliconInsight

If Tessent Shell is unable to acquire a license when setting the context, the tool returns an error message.

- **-design_identifier *design_id***

An option used to assign an identifier to the current design.

A *design_id* must only use a combination of the following characters:

- Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
- Numeric. For example: “123”.
- Underscore. For example: “A_1”.

For dft context, if this option is unspecified, the identifier defaults to “rtl” in the -rtl context, and to “gate” in the -no_rtl context. If you use the -design_identifier switch, then you must specify a *design_id* value. In the case of an empty string, the design identifier will fall back

to default value, which is gate or rtl in dft context or an empty string in the case of patterns context.

For patterns context, if this option is unspecified, the identifier defaults to either the value you previously set using `set_context` or to a null string if it was never set.

You only need to specify this option in the dft context when you are performing a second insertion pass on the same design. When you use the `process_dft_specification` command to insert all instruments in one insertion pass, the default value is adequate. In the dft context, the tool generates an error if module names matching `current_design_design_id_tessent_*` are found already instantiated into the design during transition to analysis mode or when running the `process_dft_specification` command; this indicates that you are performing a second insertion pass into that same design and that a new design identifier value needs to be specified.

- `-force`

An optional switch used to force Tessent Shell into the specified context.

Examples

Example 1

The following example demonstrates setting the context to dft. In this case, one of the dft licenses is checked out if one is not already checked out.

```
set_context dft -no_rtl
```

Example 2

The following example demonstrates setting the context to patterns -scan and requesting a TestKompress license. If the `-license` option is not specified, a FastScan license is first checked out if one is available. After that, a check is performed to determine whether LBIST or EDT modes are used; if they are, the Tessent FastScan feature is released and either the Tessent TestKompress or Tessent LBIST license is checked out, as appropriate.

```
set_context patterns -scan -license TestKompress
```

Related Topics

[report_context](#)

[get_context](#)

set_core_instance_parameters

Context: patterns -scan

Mode: setup

Changes the parameters of instruments previously added with the add_core_instances command.

Usage

```
set_core_instance_parameters {-instrument_type {edt | occ | lpct | lbist}  
| -cores core_names | -modules module_objects | -instances instance_objects }  
{-parameter_values parameter_list} [-silent]
```

Description

The instruments that the tool has extracted (that is, are a part of a core instance) have a limited set of parameters that can be overridden. This command is used to change these parameters.

Arguments

- **-instrument_type edt | occ | lpct | lbist**

A required switch and literal pair that changes all instances of a given type. Choose one from the following:

- edt
- occ
- lpct
- lbist

- **-cores core_names**

A required switch and string pair that changes the specified instrument parameters for the specified **core_names**, which is a Tcl list of one or more core names, or a collection of one or more core names.

- **-modules module_objects**

A required switch and string pair that changes the specified instrument parameters for the specified **module_objects**, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-instances instance_objects**

A required switch and string pair that changes the specified instrument parameters for the specified **instance_objects**, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-parameter_values *parameter_list***

A required switch and list that specify the instrument configuration parameter values to change. Refer to [Table 3-5](#) in the documentation for [add_core_instances](#) for a list of available instrument parameters.

- [-silent]

An optional switch that specifies to suppress warning messages.

Related Topics

[add_core_instances](#)

[import_scan_mode](#)

[report_core_instance_parameters](#)

[set_current_design](#)

Context: all contexts

Mode: setup, insertion

Specifies the top level of the design for all subsequent commands until reset by another execution of this command.

Usage

```
set_current_design [top_design [-icl_module icl_module]] [-verbose] [-in_library library]  
[-show_elaboration_warnings]
```

Description

Specifies the top level of the design for all subsequent commands until reset by another execution of this command.

When you issue this command without an argument, the tool automatically finds the top level of the design if only one exists. If the tool finds more than one top, the tool presents a list of the possible top levels; you must reissue the command and specify one of the top levels listed. Note, if you issue this command without an argument and a design is already elaborated, the same design is systematically re-elaborated.

Note that parameterized views of modules are only available to introspection if the parameterized module is instantiated below the current design. You can only set the current design to a module with default parameters, or to a non-parameterized module.

If you want to read additional designs or libraries after setting a top module, you must again issue the `set_current_design` command to re-elaborate the design before you can operate on the design.

If you have operated on the design or have read a flat model, you cannot read additional designs or libraries, or set a new top design without either deleting the current settings using the `delete_design` command or using the “`read_verilog -force`” or “`read_cell_library -force`” command.

When you use the command:

```
set_current_design [get_module M1 -in_library L1]
```

you are able to specify the logical library in which the module M1 exists. When using a `module_name` as the argument to the `set_current_design` command, the `-in_library` option is used to specify in which library to look for module name `module_name`. When the top level is an architecture instead of a module, the `get_module` command cannot be used to refer to it and you must refer to it only by name. In this case, you must use the `-in_library` switch if the architecture is not in the default logical library.

In the dft and “patterns -ijtag” context, the `set_current_design` command is used to set the current design even if the design only has an ICL view. When `set_current_design` executes in these contexts, if a loaded ICL module exists whose name matches the specified current design name, the tool sets the context to ijtag retargeting; if no loaded ICL module exists whose name matches the specified current design name, the tool enables the ICL extraction mode.

In the IJTAG retargeting context, if a design module is not already loaded, the tool creates a design module with the ports found in the ICL module. If a design module is already loaded, the tool verifies that the module contains all of the ports defined in the ICL module. If a file name `design_name.pdl` is found beside the file that contains the top level ICL module, the PDL file is automatically sourced as described in “[dft_inserted_designs](#)” on page 3768.

The tool writes out patterns that include the list of ports found within the design module as well as those in the ICL module. If you plan to use a `test_setup` procedure that interacts with ports that are in the design module but not described in the ICL module, you must load the design view and not allow it to be created from the ICL module.

Ports (and their attributes) are not deleted by the `set_current_design` command when their module is no longer in the elaboration tree. Instead, the tool stores this information, which becomes accessible when their module re-appears in the elaboration tree, for example, due to another issuance of the `set_current_design` command.

Elaborated Designs and Importing Generated Clocks

When in the DFT context with no sub-context, added clocks are rule checked when `set_dft_specification_requirements -memory_test` and/or `-logic_test` are set to on.

When `process_dft_specification` is called, the rule-checked clock definitions are stored into the TSDB in the following location within the `dft_inserted_designs` directory:

```
design_name_design_id.dft_inserted_design_design/design_name.tcd
```

In successive DFT insertion passes when the `set_current_design` command is called in the DFT context with no sub-context, the clocks are automatically restored. They are also automatically restored if the `set_current_design` command is called in the patterns -ijtag context. In contrast, the clocks are not automatically restored in the other contexts.

In the patterns -scan context, you use the `import_scan_mode` command followed by the `import_clocks` command to import the clocks. The `import_scan_mode` command only works if you used Tesson Scan for scan insertion, and the `import_clocks` command only works if you ran the DFT step described above that resulted in the clocks being stored in the TSDB.

Arguments

- *top_design*

An optional string that specifies the name of the top-level module in the design. When the module name is provided, the `set_current_design` command looks for the module in the

default library. If you want to make a module from a different library the current design, you can use the [get_modules](#) command to return the module and then pass the module object directly to set_current_design as follows:

```
set_current_design \
    [get_modules module_name -in_library library_name]
```

- **-icl_module *icl_module***

An optional switch and string pair that specifies the name of an already loaded ICL module to uniquify and associate with the current design being set. This switch must be used in the first insertion pass after synthesis when your RTL design is parameterized and multiple netlists are created for the design by your synthesis tool. It specifies the ICL module in the RTL view that maps to the top module of the netlist being elaborated.

As part of the first DFT insertion pass for a netlist generated by synthesis, the ICL must be uniquified in order to reconcile the naming of the netlist top module with that of the top ICL module. The set_current_design command invoked as “set_current_design *design_name* -icl_module *icl_module_name*” performs this uniquification. It updates the name of the ICL module to match the name of the top module of the netlist (*design_name*). All references to the pre-uniquified module are updated with the new name in the .icl, .tcd, and .tsdb_info files in the ./dft_inserted_designs/<block_name>.dft_inserted_design directory. In addition, the attribute tessent_original_module_name is added to the wrapper for the unqualified ICL module in the .icl file. The value of that attribute is the original name of the top module.

Further, when you perform the DFT insertion on the netlist at the parent block level, the ICL hierarchy in the .icl file in the

./dft_inserted_designs/<parent_block_name>.dft_inserted_design directory is updated to reflect the child block top modules in the gate view. The instance properties and the child ICL module names are all updated to reflect the child netlist names. The ChildBlockModules wrappers in the .tcd and .tsdb_info files are updated as well.

- **-in_library *library***

An optional switch and value pair that specifies to restrict the search to a specified library. The *library* value must be a single string corresponding to one of the logical library names defined with the [set_logical_design_libraries](#) command. When the -in_library option is not specified, the tool will only look for modules in the default library. Use this switch when the specified module or top level architecture name is not in the default library. This option is only allowed in the rtl context.

- **-verbose**

An optional switch that automatically reports each ICL file read by design module matching. For every ICL file read, the following messages are returned:

```
// Reading icl file filename
// Reading icl file filename
```

- **-show_elaboration_warnings**

An optional switch that displays details of netlist and library warnings.

Examples

Example 1

The following example sets the top design to block_1:

```
set_current_design block_1
// Note: Top design is 'block_1'.
```

Example 2

The following example finds the top level of the design (block_2) and sets it as the top. This use model only works in the no_rtl context and requires that all modules currently loaded are instantiated below a single module.

```
set_current_design
// Note: Top design is 'block_2'.
```

Example 3

The following example loads a mixed-language design in the DFT context in RTL mode.

```
SETUP> set_context dft -rtl
# Specify library1 and work as the two active libraries
SETUP> set_logical_design_libraries -logical_library_list {library1 work} \
-default_library work
SETUP> read_vhdl core*.vhdl -in_library library1 -format 93
# Set the current_design to core1 from library1
SETUP> set_current_design [get_modules core1 -in_library library1]
// Note: Top design is 'core1'.
```

Example 4

In the following example the parameterized design top_core_pv1 has two netlists after synthesis:

- top_core_pv1_1
- top_core_pv1_2

You load the design top_core_pv1_1 in preparation for the first gate_level insertion pass. When the design is elaborated using the set_current_design command, you need to specify the name of the top ICL module which existed in the RTL view as the value of the -icl_module option. The ICL module 'top_core_pv1' is renamed to 'top_core_pv1_1' and the original name is recorded in the attribute tesson_original_module_name.

```
SETUP> set_context dft -no_rtl -design_id gate1
SETUP> open_tsdb top_core_pv1.tsdb
SETUP> read_verilog ./synth/top_core_pv1/gates.top_core_pv1_1.v
SETUP> read_design top_core_pv1 -design_id rtl -no_hdl
SETUP> set_current_design top_core_pv1_1 -icl_module top_core_pv1
```

Related Topics

[delete_cell_library](#)
[delete_design](#)
[read_flat_model](#)
[read_cell_library](#)
[read_verilog](#)
[set_cell_library_options](#)
[set_design_include_directories](#)
[set_design_macros](#)

set_current_edt_block

Context: dft -edt, patterns -scan, patterns -scan_retargeting,
patterns -scan_diagnosis

Mode: setup, analysis

Directs the tool to apply certain subsequent commands to a particular EDT block only.

Usage

`set_current_edt_block block_name`

Description

Directs the tool to apply certain subsequent commands to a particular EDT block only.

The `set_current_edt_block` command restricts the applicability of certain subsequent commands to the EDT block having the specified *block_name*. This is referred to as setting or changing the EDT context and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the [delete_edt_blocks](#) command, or you change the context to a different EDT block using the [add_edt_blocks](#) command or another `set_current_edt_block` command.

Use this command when you want to define or redefine the attributes of a specific EDT block using context-sensitive commands. The `add_scan_chains` command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (`set_system_mode`, `create_patterns` and `report_statistics` for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block (“[set_edt_options -channels](#)” for example) will return an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- ***block_name***

A required string that specifies the name of the EDT block you wish to set as the current EDT block for subsequent context-sensitive commands. The *block_name* must be one you defined previously using the `add_edt_blocks` command. To view a list of the current defined block names, enter “`report_edt_blocks`”.

Examples

Example 1

The following example lists current user-defined block names, then changes the EDT context to the EDT block named “my_core1_block”:

```
report_edt_blocks
// my_core1_block
// my_core2_block
// my_core3_block
// my_core4_block      (current block)

set_current_edt_block my_core1_block
report_edt_blocks
// my_core1_block      (current block)
// my_core2_block
// my_core3_block
// my_core4_block
```

Example 2

The following example uses the `set_current_edt_block` command to force the context-sensitive `report_edt_configuration` command to report on specific EDT blocks:

```
set_current_edt_block my_core2_block
report_edt_configuration

//
// EDT block "my_core2_block"
// -----
// IP version:                  3
// External scan channels:    3
// Decompressor size:          10
// Clocking:                   edge-sensitive
//
// Note: Only current EDT block "my_core2_block" is reported. Use "report
//       edt configuration -all_blocks" to report the configurations of all
//       blocks.

set_current_edt_block my_core4_block
report_edt_configuration

//
// EDT block "my_core4_block"
// -----
// IP version:                  3
// External scan channels:    1
// Decompressor size:          8
// Clocking:                   edge-sensitive
//
// Note: Only current EDT block "my_core4_block" is reported. Use "report
//       edt configuration -all_blocks" to report the configurations of all
//       blocks.
```

Related Topics

[add_edt_blocks](#)

[delete_edt_blocks](#)

[report_edt_blocks](#)

set_current_edt_configuration

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the [add_edt_configurations](#) command.

Sets a particular compression configuration for the tool to use within the current design block.

Usage

`set_current_edt_configuration config_name`

Description

Sets a particular compression configuration for the tool to use within the current design block.

This is necessary so that subsequent commands can be applied to define or redefine the attributes of the specific compression configuration.

For more information about compression configurations, see the “[Defining Dual Compression Configurations](#)” section of the *Tessent TestKompress User’s Manual*.

Arguments

- ***config_name***

A required string that specifies the name of a compression configuration. The *config_name* is the name specified when the compression configurations were defined. Use the `report_edt_configurations` command to display a list of the currently defined configuration names.

Examples

The following example sets the current EDT block, *my_core1_block*, and the compression configuration, *system_test*. Then, it redefines the input and output channels and reports the new configurations.

```
set_current_edt_block my_core1_block
report_edt_configuration

// my_core1_block
// IP version: 4
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10] [system_test : 4]
// Output channels: [manufacturing_test : 5] [system_test : 2]
// Bypass chains: 5

set_current_edt_configuration system_test
set_edt_options -input_channels 3 -output_channels 3
report_edt_configuration
```

```
// my_core1_block
// IP version: 4
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10] [system_test : 3]
// Output channels: [manufacturing_test : 5] [system_test : 3]
// Bypass chains: 5
```

Related Topics

[add_edt_configurations](#)

[delete_edt_configurations](#)

[report_edt_configurations](#)

set_current_mode

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting,
patterns -failure_mapping

Mode: setup

Specifies the current test mode in pattern retargeting and core mapping flows.

Usage

For -scan and -scan_diagnosis:

```
set_current_mode [mode_name] [-type mode_type]
```

For -scan_retargeting and -failure_mapping:

```
set_current_mode mode_name
```

Description

The set_current_mode command specifies the current test mode in pattern retargeting and core mapping flows. The different types of test modes (unwrapped, internal, external, and retargeting) imply different operating behaviors of the scan chains. The command specifies that the design's scan configuration is set up for a particular test mode, mainly for the purpose of generating retargetable patterns for a wrapped core, but you can also use the command in the “core mapping for ATPG” process. Depending on the mode type, the ATPG tool may apply certain constraints automatically (as described under “-type” below). For all mode types, the mode name and mode type are captured in the Tesson Core Description (TCD) file, which is used for pattern retargeting or core mapping.

In addition to the three test modes described under “-type” below, there is a fourth test mode called “retargeting,” which you cannot set with the -type switch and is the only test mode available in the “patterns -scan_retargeting” and “patterns -failure_mapping” contexts. The “retargeting” test mode provides access to lower-level cores for the purpose of retargeting patterns. Because “retargeting” is the only available test mode, it is not necessary to use the set_current_mode command in these contexts. The only situation in which you need to use the set_current_mode command is when you are in the “patterns -scan” or “patterns -scan_diagnosis” contexts and you want to do pattern retargeting.

If you have explicitly set the type and then use set_current_mode, without specifying -type, the tool will leave the type as it was set, rather than reset it to the context default.

For more information about the mapping process and TCD files, refer to “[Core Mapping for ATPG Process Overview](#)” in the *Tesson Scan and ATPG User’s Manual*. For more information about pattern retargeting, refer to “[Scan Pattern Retargeting](#)” in the *Tesson Scan and ATPG User’s Manual*.

Arguments

- *mode_name*

An optional or required literal that specifies a user-defined name for the mode. If you don't specify this argument, the *mode_name* argument defaults to the value of *mode_type*. The *mode_name* is optional for the *-scan* and *-scan_diagnosis* sub-contexts and is required for the *-scan_retargeting* and *-failure_mapping* sub-contexts.

- *-type mode_type (“patterns -scan” and “patterns -scan_diagnosis” only)*

An optional switch that specifies the test mode type for the tool when generating retargetable patterns. Valid options are:

unwrapped — Specifies the current default test mode in which ATPG runs in the “patterns -scan” context. This mode does not support core isolation or hierarchical test, so use this mode when generating patterns for designs that do not contain wrapper chains.

internal — Specifies the internal test mode of a wrapped core in which the tool generates patterns while the core is isolated from the rest of the design. Due to this isolation, those patterns are independent of circuitry outside of the core's wrapper chain and can therefore later be retargeted from the core level to a higher level of the design. When you specify this option, the tool automatically adds CX constraints to all primary inputs and masks all primary outputs during design rule checking and pattern generation. However, exceptions include primary inputs already constrained to 1 or 0, any clock, and scan chain or scan channel inputs/outputs. For more information, refer to “[Core-Level Pattern Generation](#)” in the *Tessent Scan and ATPG User's Manual*.

When you run ATPG in the internal type mode, primary inputs (PIs) and primary outputs (POs) are not valid fault sites when the core is embedded at the next level up, which creates inconsistencies between fault accounting at the core level and those at the top. The tool behavior is to not fault PIs and POs in this situation.

external— Specifies the external test mode of a core in which the tool generates patterns for the interconnect circuitry outside of the wrapper chain. You use the external mode after instantiating the core or a graybox representation in the design. For more information, refer to “[Generating Patterns in External Mode](#)” in the *Tessent Scan and ATPG User's Manual*. Note that using external mode does not change constraints or masking of the core pins (as with internal mode), so external mode is identical to unwrapped mode with regard to the tool's ATPG functionality; however, unlike unwrapped mode, external mode implies that you have instantiated wrapped cores.

Examples

This example sets the mode name and mode type to “internal”.

```
set_context patterns -scan
set_current_mode -type internal
```

Related Topics

[get_current_mode](#)

set_current_silicon_insight_setup

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Sets the currently selected setup in the Tesson SiliconInsight Setup Specification.

Usage

`set_current_silicon_insight_setup setup_type`

Description

Each adaptor supports offline mode as indicated by the “_offline” suffix. Use this mode for testing that the patterns run well with the adaptor.

This command only affects the current session. If you want to save a change to the setup specification so that you can read it in using the `read_config_data` command in a subsequent session, you must use the `write_config_data` command.

Arguments

- *setup_type*

A required string that specifies one of the supported Tesson SiliconInsight setup specification types. Options are:

NoTester(batch_cdp)
Sid(jtagkey)
Sid(jtagkey_offline)
Sid(flyswatter2)
Sid(flyswatter2_offline)
Sid(olimex_arm_usb_ocd)
Sid(olimex_arm_usb_ocd_offline)
Sid(olimex_arm_usb_ocd_h)
Sid(olimex_arm_usb_ocd_h_offline)
Sid(opalKellyXem6310)
Sid(opalKellyXem6310_offline)
Sid(signalyzer)
Sid(signalyzer_offline)
Sid(signalyzerSHA40)
Sid(signalyzerSHA40_offline)
Sid(signalyzer40)
Sid(signalyzer40_offline)

Sid(signalyzerH4)
Sid(signalyzerH4_offline)
Sid(signalyzerSP)
Sid(signalyzerSP_offline)
Sid(simdut)
Ate(ate1)

Examples

set_current_silicon_insight_setup Sid(simdut)

set_current_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Sets the current simulation context.

Usage

set_current_simulation_context *context_name*

Description

The set_current_simulation_context command sets the simulation context for later analysis and introspection. [Table 6-7](#) lists the predefined simulation contexts.

Table 6-7. Predefined Simulation Contexts

Simulation Context	Description
stable_after_setup	Specifies the values that are constant in the period following the end of the test_setup procedure and the beginning of the test_end procedure. For scan patterns, the value is constant throughout scan and capture cycles. For IJTAG patterns, the value is constant throughout any IJTAG operations.
stable_capture	Specifies the values that are constant throughout the capture procedures.
stable_load_unload	Specifies the values that are constant throughout the load_unload cycles.
stable_shift	Specifies the values that are constant throughout the shift cycles.
tie_value	Specifies the values that are constant due to tied nodes in the design. For example a net assigned to 1'b0 will have a tie value of 0. Primary inputs constrained with the “ add_input_constraints -ct0 or -ct1 ” switches as also considered tied and will be reported as constant in this simulation context.

Arguments

- *context_name*

A required literal that specifies the simulation context to make current. You can specify any of the simulation contexts listed in [Table 6-7](#) or any user-defined simulation contexts that have been added. By default, the simulation context is stable_after_setup.

Examples

The following example sets the current simulation context to stable_shift:

```
> set_current_simulation_context stable_shift
```

Related Topics

[get_current_simulation_context](#)
[get_simulation_context_list](#)
[report_simulation_contexts](#)

set_decision_order

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies how the ATPG selects gate input or output nodes for evaluation during ATPG.

Usage

`set_decision_order -NORandom | -Random`

Description

Specifies how the ATPG selects gate input or output nodes for evaluation during ATPG.

The `set_decision_order` command specifies whether ATPG chooses a gate input or output for evaluation randomly, or by its ease of evaluation. Ease of evaluation for a gate input depends on how easy it is for the tool to control the input. For a gate output, ease of evaluation is related to how easily the tool can observe the output. A nonrandom decision order causes the tool to evaluate the “easiest” gates first.

Arguments

- **-NORandom**

A required switch that causes the ATPG to choose the easiest evaluation when faced with a decision about which gate input or output to evaluate next. This is the default unless you are using the [create_patterns](#) command, which always uses a random decision order.

- **-Random**

A required switch that causes ATPG, when faced with a decision about which gate input or output to evaluate next, to make random decisions rather than choosing the easiest evaluation. This causes variation between patterns, tending to give a more compact vector set at the risk of creating more ATPG aborts.

Note

 In some cases, depending on circuit topology, changing the order in which the tool makes decisions during ATPG may improve performance. If large numbers of faults are being aborted, for example, you may eliminate or reduce aborted faults by setting the decision order to “random” without changing the abort limit.

Related Topics

[create_patterns](#)

set_dedicated_wrapper_cell_options

Context: dft -scan

Mode: setup

This command controls the inferring of the dedicated wrapper cells on all or the specified primary IOs.

Usage

```
set_dedicated_wrapper_cell_options [on | off | auto]
    [-ports port_spec]
    [-input_module_name module_spec]
    [-output_module_name module_spec]
    [-clock_source port_pin_spec]
```

Description

This command controls the inferring of the dedicated wrapper cells on all or the specified primary IOs, that is, it defines how each port of the current design is to be handled during wrapper analysis. It is an incremental command and can be used multiple times to generate a desired superset of options.

Note

 During wrapper analysis, the tool will check whether a PI is driving the Set/Reset ports of a flop encountered during the forward tracing, and if so, it will infer a dedicated wrapper cell on this PI.

Arguments

- **on | off | auto**

Ports that are excluded from wrapper analysis will not be affected by this command. If excluded ports are specified with the -ports switch or if the -ports switch is not specified and there are excluded ports in the design you will see the following warning:

```
// Warning: Ignoring excluded port(s) <port1>, <port2>, <port3> ...
//           To include these ports in wrapper analysis please use:
//           set_wrapper_analysis_options -clear_excluded_ports
```

An optional switch that specifies whether to auto infer the dedicated wrapper cells on all or the specified primary IOs, or to have the dedicated wrapper cells added to all or the specified primary IOs, or to have all or the specified primary IOs excluded from having the dedicated wrapper cells added to them:

Undriven outputs receive dedicated wrapper cells based on the configurations of “set_wrapper_analysis_options –register_undriven_output on|off” and will not be affected by global on|off|auto settings. However, specifying a specific undriven port (or group of undriven ports) with “-ports {...}” will override the –register_undriven_output settings for that/those specific port(s) as seen in Example 1 below.

- on — specifies to add a dedicated wrapper cell for the given ports. The “on” state is used to automatically register all primary IOs specified with the -ports switch (including undriven outputs), or to register all primary IOs (with the exception of undriven outputs) when the -ports specification is not provided.
- off — specifies to not allow inferring a dedicated wrapper cell for the given ports. The “off” state is used to exclude from the registration either the primary IOs specified with the -ports switch (including undriven output), or all primary IOs (with the exception of undriven output) when the -ports specification is not provided.

In addition, any previous mapping of clock source to wrapper cells will be cleared for the given ports (or for all ports if the -ports switch is not specified).

- auto — specifies that dedicated wrapper cells are inferred and determined by the tool during the wrapper cells analysis for the primary IOs specified with the -ports switch or all primary IOs when the -ports specification is not provided . This is the default.
- **-ports *port_spec***
An optional switch and string that specifies the ports for which the current setting applies to. When this switch is not specified, the dedicated wrapper cell inferring setting is applicable to all primary Input and Output ports.
port_spec can be a Tcl list of one or more ports or a collection of one or more port objects.
- **-input_module_name *module_spec***
An optional switch and string that specifies a library model to be used to instantiate a dedicated Input wrapper cell.
module_spec can be the name of a library model or a collection containing a library model object.
- **-output_module_name *module_spec***
An optional switch and string that specifies a library model to be used to instantiate a dedicated Output wrapper cell.
module_spec can be the name of a library model or a collection containing a single library model object.
- **-clock_source *port_pin_spec***
An optional switch and string that specifies an internal pin or top level port to be used as the clock source for the inserted dedicated wrapper cells. This clock source will be applied only to the dedicated wrapper cells of the specified primary IOs when the -ports switch is specified, or to the dedicated wrapper cells of all primary IOs when the -ports switch is not specified.
port_pin_spec can be the name of a pin/port, or a collection containing a single pin/port object.

This switch is ignored when combined with “off”, and the tool will issue the following error message:

```
// Error: -clock_source has no effect when dedicated wrapper cells
are off (disabled).
```

Note

For more information, see [Clock Selection for Dedicated Wrapper Cells](#) in the *Tessent Scan and ATPG User’s Manual*.

Examples

Example 1

In this example, all driven Primary IOs will receive dedicated wrapper cells. Undriven IOs will not receive dedicated wrapper cells:

```
set_dedicated_wrapper_cells on
```

Example 2

In this example, all Primary IOs will receive dedicated wrapper cells, including undriven IOs.

```
set_dedicated_wrapper_cells on
set_wrapper_analysis_options –register_undriven_output on
```

Example 3

In the following example:

Ports A, B, C, D, excluded_1 are driven.

Ports undriven_1, undriven_2 are undriven.

Ports A, D, undriven_1 will receive dedicated wrapper cells.

Ports B, C, undriven_2, excluded_1 will not receive dedicated wrapper cells.

```
set_wrapper_analysis_options –exclude_ports excluded_1
set_dedicated_wrapper_cells on
set_dedicated_wrapper_cells off –ports {A B C}
set_dedicated_wrapper_cells on –ports {A undriven_1}
```

Related Topics

[set_wrapper_analysis_options](#)

set_defaults_value

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Sets the default value of a property in one of the DefaultsSpecification (policy) wrappers.

Usage

```
set_defaults_value property_path_name {value | -unspecify} [-policy policy]
```

Description

This command has access to and sets default values for the properties of the sub-wrappers in the top [DefaultsSpecification\(<policy>\)](#) to match your preferences and requirements. The [DefaultsSpecification\(<policy>\)](#) sub-wrappers are [DftSpecification](#) and [PatternsSpecification](#).

The set_defaults_value command automatically manages the priority of the <policy> and sets the value of the highest priority by default. The policies, in order of highest priority to lowest are: user, group, and company.

Note

 The values set using this command impact the default values used by the [create_dft_specification](#) command. They do not impact existing dft specifications.

Use the command “[report_config_syntax](#) DefaultsSpecification” to see all the available properties you can set. You can use the -level 1 and -wrapper_only switches to see the next level of sub wrappers and add the wrapper name you care about to the path to limit the size of the report as follows:

```
> report_config_syntax DefaultsSpecification/DftSpecification -level 1 \
  -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
    }
    BoundaryScan {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
  }
}

> report_config_syntax DefaultsSpecification -level 1 -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  PatternsSpecification {
  }
  DftSpecification {
  }
}
```

```
> report_config_syntax DefaultsSpecification/DftSpecification -level 1 \
-wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
    }
    BoundaryScan {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
  }
}

> report_config_syntax DefaultsSpecification/DftSpecification/\
IjtagNetwork -level 1 -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
      ImplementationOptions {
      }
      HostScanInterface {
      }
    }
  }
}

SETUP> report_config_syntax DefaultsSpecification/DftSpecification/\
IjtagNetwork/ImplementationOptions -level 1
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
      ImplementationOptions {
        scan_path_retiming : <string> ; // legal : (latch) flop
      }
    }
  }
}
```

Arguments

- *property_path_name*

A required string that specifies a configuration pathname *without* the DefaultsSpecification prefix. For the DftSpecification, this path starts with “DftSpecification/”prefix. For the PatternsSpecification, this path starts with “PatternsSpecification/”. For example:

```
set_defaults_value \
DftSpecification/MemoryBist/ControllerOptions/gate_bist_clock_in_functional_mode off
```

- *value*

An optional value that specifies the value of the specified property.

- **-unspecify**

An optional value that unspecifies the property instead of setting its value. Use this option to remove a property setting for a policy.

- **-policy *policy***

An optional switch and value pair that specifies for which DefaultsSpecification (policy) wrapper the property is to be set or unspecified. The available policies, in order from highest priority to lowest, are: user, group, and company. When this switch is not specified, the user policy is used.

Examples

Example 1

The following example sets the rtl_extension property in the company DftSpecification policy and unspecifies the same property in the user policy.

```
set_defaults_value DftSpecification/rtl_extension vb -policy company
```

Unspecify the rtl_extension property in the DftSpecification user policy.

```
set_defaults_value DftSpecification/rtl_extension unspecify -policy user
```

Because the user policy for this property was just unspecified and assuming the group policy for this property was not set, it will return vb:

```
get_defaults_value DftSpecification/rtl_extension
```

Example 2

This example shows using set_defaults_value and get_defaults_value that, when automatically resolving the priority, the default value of the property being set is accessed.

```
get_defaults_value PatternsSpecification/MaufacturingOptions/manufacturing_patterns_format
```

The command returns STIL, the default, because the property, in this case, is not set for any of the PatternsSpecification priorities.

This command sets the value of the “group” policy for this property to Wgl:

```
set_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_format Wgl \
-priority group
```

This command returns the default value for the PatternsSpecification/ManufacturingOptions/manufacturing_patterns_format property:

```
get_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_format
```

The command returns wgl because only the PatternsSpecification/ManufacturingOptions/manufacturing_patterns_format group priority property has been set. The get_defaults_value command first looks for a value for the property in the user priority and if not set, it looks in the group priority and finds, in this case, wgl.

This command demonstrates that the tool uses the default value for the property when it has not been set:

```
get_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_format \
-policies company
```

This returns STIL, the default for the “company” policy,

Related Topics

[get_defaults_value](#)

[create_dft_specification](#)

set_design_include_directories

Context: unspecified, all contexts

Mode: setup

Specifies one or more directory paths that contain `include files for use by subsequent read_verilog commands.

Usage

```
set_design_include_directories include_dir ... | [-Clear]
```

Description

Specifies one or more directory paths that contain `include files for use by subsequent read_verilog commands.

This command provides equivalent functionality to the –Incdir invocation switch. If you issue this command multiple times, the last set of include directories is used. When you issue this command without arguments, the command reports the list of include directories.

Each *include_directory* must be relative to the current tool invocation directory or an absolute directory pathname. Include files are searched for in the following order of precedence:

1. Absolute pathnames specified by `include directives in the Verilog design.
2. Directories specified with the command.

For relative `include file targets, directories are searched in the order they are listed, left to right, in the command. If identically named files exist in multiple directories, the first file found is used and the others are ignored.

3. Directory where the Verilog design file is located.
4. Directory in which you invoked the tool.

Arguments

- *include_dir*
A string that specifies one or more directory paths.
- -Clear
An optional switch that removes the current list of include directories.

Examples

The following commands specify that the tool search for `include files contained in ..dir1 when loading the top.v module:

```
set_design_include_directories ./dir1
read_verilog top.v
```

Related Topics

[delete_design](#)
[read_verilog](#)
[set_current_design](#)
[set_design_macros](#)
[read_cell_library](#)
[read_flat_model](#)

[set_design_level](#)

Context: dft, patterns

Mode: setup, insertion

Prerequisites: You can set the level can only after the design is elaborated.

Specifies the level of the current design.

Usage

```
set_design_level {chip | physical_block | sub_block}
```

Description

Specifies the level of the current design.

You must specify the design level with this command when you are in the “dft -rtl” or “dft -no_rtl” contexts and want to change to analysis system mode. There is no default setting, so you must specify the appropriate level because this setting affects the [create_dft_specification](#) and [process_dft_specification](#) commands, as well as some design rules checks that the tool performs when changing to the analysis system mode.

If you issue the [extract_icl](#) command immediately after having issued the [process_dft_specification](#) command, and if you specified the design level as chip or physical_block while in dft context, the tool uses the specified design level during [extract_icl](#) to set the [tessent_is_physical_module](#) attribute to true on the top ICL module. If you did not run the [extract_icl](#) command in the same tool session as the dft step, you must set the design level prior to issuing the [extract_icl](#) command.

Understanding the difference between a physical block and a sub-block is critical. A physical block is synthesized and laid out as a separate entity. When a physical block is instantiated below another physical block, the netlist view of the child physical block is not loaded into the synthesis or layout tool when synthesizing the parent physical block. Instead, a timing model for the child physical block is used. This means the SDC created for the parent physical block can only refer to pins of the child physical block. It is not possible to reference any elements within the child physical block. You cannot on the other hand refer to pins of a sub-block using -to or -from when synthesizing it within its parent physical block. Instead you refer to the elements inside the sub-block. Another major difference is the fact that the netlist of the sub-block will typically only exist as part of the parent physical netlist. You will do scan insertion from the physical block boundary and insert the scan chains into the sub-block at the same time. If the lower block is a physical block, you insert the chains inside the lower physical block first and then you connect to those scan chains when inserting scan chains in the parent physical block.

In the RTL DFT flow, the sub-block is treated pretty much the same way as if it was a physical block. If you want to use the [read_def](#) command to extract the placement information of the memories to build one memoryBIST controller per physical cluster, you have to use the “-instance_of_current_design” switch in the [read_def](#) command if you are inserting memoryBIST in a sub-block. You only can get a DEF file for a physical block. By definition,

you cannot get a DEF file for a sub-block because it is synthesized within the parent physical block. You use the DEF file of the parent physical block and use the “-instance_of_current_design” switch to specify what is the instance path of your sub-block within the parent physical block DEF file.

During signoff simulation, the simulation test benches are separated per physical block to optimize performance during simulation. See the [create_patterns_specification](#) command description and the [simulate_instruments_in_lower_physical_instances](#) property in the DefaultsSpecification/PatternsSpecification/SignOffOptions wrapper for more information about the signoff simulation. If you have sub-block modules, they are simulated as part of their parent physical block because by the time you get to gate-level simulation, you will only have a netlist and an SDF file per physical block as the sub-blocks are synthesized within their parent physical block.

When you use [read_design](#) to load a sub-block inside the tool, the [is_hard_module](#) Module-level attribute is automatically inferred to “true” on the sub-block modules if they are not the current_design and the -rtl context is used. When you insert DFT elements inside the current design, you are only allowed to make connections to the pins of the sub-block modules. If you need to add something inside the sub-block, you have to make the sub-block module the current design and do the additions at that level. You also extract the ICL file for the sub-block at that level. Remember that ICL extraction in the parent physical region is performed by re-using the ICL description of the sub-block and tracing outward from the pins of the sub-block instances. If edits were done in the sub-block module from the parent level, the changes would not be reflected in the extracted ICL. When you reach the gate-level DFT flow, as was described above, the scan insertion is performed from the parent physical block level and the scan chains are inserted inside the sub-block at the same time as in the parent physical block. Inserting scan chains and test points in the netlist has no impact on the ICL network. You do not need to re-run ICL extraction post-synthesis unless you are inserting IJTAG nodes after synthesis. Just like the rest of your functional logic, the equivalence between the pre- and post-synthesis views is verified using formal verification tools. The ICL network is functional logic to formal verification tools.

Arguments

- **chip | physical_block | sub_block**

A required literal that specifies the design level:

chip — The design is intended to be used at the chip level. This implies that all ports have a pad buffer and a TAP is inserted during the execution of the [create_dft_specification](#) command. A chip is by definition also a physical block. You can use value “top” as a synonym to “chip”.

physical_block — The design is intended to be used for blocks that are synthesized and laid out as independent blocks. There are several implication when defining a module as a physical block vs a sub-block. Those are described above. Make sure to specify the right value.

sub_block — The design is intended to be used for blocks that are synthesized and laid out as part of a parent physical block. There are several implication when defining a module as a physical block vs a sub-block. Those are described above. Make sure to specify the right value.

Note that there is no default setting, so you must specify one of these design levels.

Examples

The following example sets the level of the current design to `physical_block`.

`set_design_level physical_block`

Related Topics

[get_design_level](#)

[set_current_design](#)

set_design_macros

Context: unspecified, all contexts

Mode: setup

Specifies one or more defined macros for use by subsequent commands.

Usage

```
set_design_macros macro_name[=macro_value] ... | [-Clear]
```

Description

Specifies one or more defined macros for use by subsequent commands.

This gives you the convenience of using macros rather than having to edit the netlist to explicitly add the `define statements to control which parts of `ifdef statements are compiled. This command provides functionality equivalent to the –Define invocation switch.

If you issue the command multiple times, then the last set of macros is used. When you issue the command without arguments, the command reports the list of defined macros, including the default macros MGC_TESSENT and SYNTHESIS.

Arguments

- *macro_name[=macro_value]*
An optional string that specifies one or more macros.
- -Clear
An optional switch that clears the current list of all defined macros.

Examples

The following commands specify that the tool use a custom macro named macro_1 set to a value of 20 when loading the top.v module:

```
set_design_macros macro_1=20
read_verilog top.v
```

Related Topics

- [delete_design](#)
- [read_verilog](#)
- [set_current_design](#)
- [read_cell_library](#)
- [read_flat_model](#)

set_design_sources

Context: unspecified, all contexts

Mode: setup

Specifies where the tool should look for the definition of undefined modules in the list of files specified by the [read_verilog](#) command.

Usage

```
set_design_sources {[ -format verilog ] {[ -V file_path ... ] [-Y dir_path ... -extensions ext ... ] [-append] } | { [-clear] } }

set_design_sources -format {icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory
| tcd_memory_cluster} {[ -search_design_load_path {on | off} ]
[-Y dir_path ... -extensions ext ...] [-append]} [-clear]

set_design_sources { -format {icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory
| tcd_memory_cluster} [-clear]}
```

Description

Specifies where the tool should look for the definition of undefined modules in the list of files specified by the [read_verilog](#) command when -format is specified as or defaults to “verilog”. It specifies where files are searched for during module matching for the other formats.

For more information about the module matching process, see the description of the [set_module_matching_options](#) command. If the tcd_scan, tcd_memory, tcd_memory_cluster, tcd_bscan, or tcd_fusebox wrappers are located in files with names that do not match the module_name of the Core wrapper, or if they do not have unique extensions for each format, use the [read_core_descriptions](#) command to load them in explicitly instead of trying to have them auto-loaded.

The -v, -y, and -extensions switches have similar behavior to standard Verilog compilers like ModelSim.

When you issue the command multiple times, the tool uses only the last set of -v/-y/-extension lists. You can set the -y/-v/-extension list independently of the -search_design_load_path option, but the -y/-v/-extension forms a set for each format.

Table 6-8 shows the default extensions for the various input formats.

Table 6-8. Default Extensions for set_design_sources

Format	Default Extension
icl	icl icl.gz
tcd_bscan	tcd_bscan lvbscan tcd_bscan.gz lvbscan.gz
tcd_scan	tcd_scan tcd_scan.gz
tcd_fusebox	tcd_fusebox tcd_fusebox.gz

Table 6-8. Default Extensions for set_design_sources (cont.)

Format	Default Extension
tcd_memory	tcd_mem_lib tcd_mem_lib.gz
tcd_memory_cluster	tcd_mem_cluster_lib tcd_mem_cluster_lib.gz

Arguments

- **-format verilog | icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster**
An optional switch that specifies which format the setting applies to. When unspecified, the format verilog is assumed.
See the description of the [set_module_matching_options](#) command for more information about the module matching process that is done using the design sources for formats other than verilog.
- **-search_design_load_path on| off**
An optional switch and literal pair that specifies whether the directory into which Verilog and VHDL design modules are loaded or searched are automatically used as valid search directories for the specified format. In cases where the tool adds the Verilog path to the search path, the tool also adds the paths used for the cell libraries. This option is on by default.
- **-V file_path**
An optional repeatable switch that specifies one or more files in which to search for definitions of undefined Verilog modules. You can specify multiple –v options, multiple –y options, or both. The order of the –v/-y files and directories specified in the command determines the search order during parsing.
- **-Y dir_path ... -extensions ext ...**
An optional repeatable switch that specifies one or more directories in which to search for definitions of undefined modules when -format is defined as or defaults to “verilog”. Any file inside the –y directory should be named as *module_name.extension* in order to be parsed. You can specify multiple –v options, multiple –y options, or both. The order of the –v/-y files and directories specified in the command determines the search order during parsing.
When you use the –y switch one or many times, you are required to also use the -extensions switch once to instruct the tool to check only the files with the specified extensions. The value is a Tcl list of one or more extension names. The tool supports compressed files by expanding the extension name or list of extension names to include those names with a .gz suffix. That is, if you specify “-extensions v”, the tool expands it to “-extensions v v.gz”. You can use this switch only once per command execution.
For formats other than Verilog, the directories are searched during the module matching process. See the description of the [set_module_matching_options](#) command for more information about the module matching process that is done using the design sources for formats other than Verilog.

- **-append**

The **-append** adds the new settings of **set_design_sources** to the previous settings of previous **set_design_sources**.

- **-clear**

An optional switch that clears the list of files and directories previously specified with the **-v** or **-y** switches. The **-clear** switch operates on only one format per command execution. If you do not use the **-format** switch, the **-clear** switch operates on Verilog files.

Examples

Example 1

To illustrate how the switches work, consider the following example:

top.v	f1.v	directory d1
module top;	module foo();	foo.v bar.v foo.sv zar.v
foo i();	endmodule	
endmodule	module bar();	
	endmodule	

When you issue these commands:

```
report_design_sources
set_design_sources -v f1.v
report_design_sources
read_verilog top.v
```

then the module foo in file f1.v is parsed, but not the module bar in f1.v.

When you issue these commands:

```
report_design_sources
set_design_sources -y d1 --extensions v
report_design_sources
read_verilog top.v
```

then the file foo.v is parsed, but not the other files in the d1 directory. Note that file foo.v is parsed but not foo.sv because the specified **--extensions** is v.

In the previous example, when you issue the following commands:

```
report_design_sources
set_design_sources -y d1 --extensions v -v f1.v
report_design_sources
read_verilog top.v
```

then foo.v in d1 directory is parsed. If the commands are:

```
report_design_sources  
set_design_sources -v f1.v -y d1 -extensions v  
report_design_sources
```

then module foo in f1.v is parsed.

Example 2

In the following example, the set_design_sources command specifies to look for undefined module definitions in the dir1, dir2, and dir3 directories and to check only those files with a v_ts or v extension.

```
set_design_sources -y {dir1 dir2 dir3} -extensions {v_ts v}
```

Related Topics

[delete_design](#)
[get_design_sources](#)
[read_verilog](#)
[report_design_sources](#)
[set_current_design](#)
[set_design_include_directories](#)
[read_cell_library](#)
[read_icl](#)

set_dft_enable_options

Context: patterns -scan

Mode: setup

Enables or disables the control or observe points.

Usage

```
set_dft_enable_options -type {xbounding_en | control_point_en | observe_point_en |  
mcp_bounding_en | test_en} -pin_name pin_name [-value {on | off}]
```

Description

Enables or disables the control or observe points.

In the hybrid TK/LBIST flow, the tool uses this command during fault simulation to specify the static enable signals inside the LogicBIST controller and during the IP creation phase to specify the creation of static TDR bits inside the LogicBIST controller that will control the static enable signals.

Note that disabling either X-bounding or MCPs for LogicBIST is not allowed since it can result in Xs being captured in the scan cells.

Disabling the control or observe points enable signal can have an impact on coverage.

Arguments

- **-type xbounding_en | control_point_en | observe_point_en | mcp_bounding_en | test_en**
A required switch and literal pair that specifies the static enable type.
- **-pin_name *pin_name***
A required switch and string pair that specifies the name of the pin to enable or disable.
- **-value {on | off}**
An optional switch and literal pair that enables (on) or disables (off) the action for the specified pin. This switch is not used during IP creation.

Examples

The following example turns off observe points for transition fault simulation.

```
set_dft_enable_options -type control_point  
-pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/control_point_en -value on  
  
set_dft_enable_options -type observe_point \  
-pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/observe_point_en -value off  
  
// Warning: Disabling 'observe_point' enable signal may have an impact  
// on coverage.
```

The following example generates TDR bits inside the LogicBIST controller and uses them to drive design pins during IP creation:

```
set_dft_enable_options -type xbounding_en -pin_name xbnd_en
set_dft_enable_options -type control_point_en \
    -pin_name [get_pins {core_i*/cp_en core_i*/obs_en}]
    # A single TDR bit controls both the control point and
    # observe point enable pins.
```

Related Topics

[set_lbist_controller_options](#)

[set_lbist_pins](#)

[set_dft_specification_requirements](#)

Context: dft

Mode: setup

Specifies the requirements to be checked when the [check_design_rules](#) command runs, and those to be included into the [DftSpecification](#) when the [create_dft_specification](#) command is run.

Usage

```
set_dft_specification_requirements [-memory_test {on | off}] [-memory_bist {auto | off}]
[-memory_bisr_chains {auto | on | off}]
[-memory_bisr_controller {auto | on | off}]
[-boundary_scan {on | off}]
[-ac_boundary_scan {auto | off}]
[-logic_test {on | off}]
[-tck_injection {on | off | internal_sources_only}]
```

Description

Specifies the requirements to be checked during the [check_design_rules](#) command and the instrument types that are to be included into the [DftSpecification](#) when the [create_dft_specification](#) command runs.

You typically only need to used the “-memory_test on” option when you want to implement memory test at a given level. When memory_test is on, memory_bist, memory_bisr_chains, and memory_bisr_controller default to auto; otherwise, they default to off. When boundary_scan is on, ac_boundary_scan defaults to auto; otherwise, it defaults to off.

These defaults mean that if the current level either instantiates blocks that contain memory BIST or itself contains memories not already connected to a memory BIST controller, the memory BIST pre-DFT DRC rules ([Scan Cell Data Rules \(D Rules\)-DFT_C5](#)) will be executed during the [check_design_rules](#) command, and the required specification will be added to the [DftSpecification](#) when the [create_dft_specification](#) command runs.

Arguments

- -memory_test on | off

An optional switch and value pair that specifies whether memory test is to be design rule checked and implemented. Specifying “-memory_test on” is a short cut to setting the three options -memory_bist, -memory_bisr_chains, and -memory_bisr_controller to auto.

- -memory_bist auto | off

An optional switch and value pair that specifies whether memory BIST clocking and, potentially, memory BIST controllers are to be considered. When set to auto, if the current design instantiates blocks containing memory BIST or itself contains memories not already connected to a memory BIST controller, then the memory BIST pre-DFT DRC rules ([Scan](#)

Cell Data Rules (D Rules)-DFT_C5) will be executed during the [check_design_rules](#) command, and the required specification will be added to the [DftSpecification](#) when the [create_dft_specification](#) command runs.

- **-memory_bisr_chains auto | on | off**

An optional switch and value pair that specifies whether memory BISR chains are to be inserted or connected in the current level. When set to auto, if the current design instantiates blocks containing memory BISR chains or itself contains repairable memories not already connected to a memory BISR register, the bisr_order_file will be validated or created during the [check_design_rules](#) command and the required specification will be added to the [DftSpecification](#) when the [create_dft_specification](#) command runs. The bisr_order_file is created taking into account physical placement when a DEF file is read in using the [read_def](#) command prior to running the [check_design_rules](#) command. You can also partition the BISR chains into power domains if you read in a UPF or CPF file using the [read_upf](#) or [read_cpf](#) command prior to running the [check_design_rules](#) command.

- **-memory_bisr_controller auto | on | off**

An optional switch and value pair that specifies whether a memory BISR controller is to be inserted in the current level. When set to auto, a controller will be inserted if the design level was specified as “chip” using the [set_design_level](#) command and either memory BISR chains exist on blocks instantiated in the current design or repairable memories exist in the current design. You can force the insertion of a controller at design levels other than chip by setting the -memory_bisr_controller option to on.

- **-boundary_scan on | off**

An optional value pair that specifies whether boundary scan is to be design rule checked and implemented. When you set -boundary_scan to on, the -ac_boundary_scan option is automatically set to auto.

If the design level is not “chip”, you can use the [set_boundary_scan_port_options](#) command to define the list of ports for which boundary scan is needed.

- **-ac_boundary_scan auto | off**

An optional switch and value pair that specifies whether ac boundary scan is to be inserted. When set to auto, AC boundary scan is inserted as soon as a pad with AC functions is present. When set to off, AC boundary scan (IEEE 1149.6) is not implemented and only IEEE 1149.1 cells are inserted.

- **-logic_test on | off**

An optional switch and value pair that specifies whether logic test is to be design rule checked and implemented. When set to on, the [DFT_C2](#), [DFT_C3](#), [DFT_C6](#), [DFT_C8](#), and [DFT_C9](#) DRC rules are run and the AsyncSetReset auto corrections implemented.

Specifying [add_dft_control_points](#) -type `async_set_reset` has no impact unless the [DFT_C9](#) rule is run and the active polarity of the asynchronous set or reset sources extracted by the DRC.

- **-tck_injection on | off | internal_sources_only**

An optional switch and value pair that specifies whether TCK muxing is to be inserted in the current level. When enabled, it will identify clock locations where TCK should be injected to drive child block ports and memories that have [use_in_memory_bist_dft_specification](#) set to “on”, or “auto” that will resolve to “on” in analysis mode. These locations will be added to the [MemoryBist](#) wrapper of the DftSpecification when the [create_dft_specification](#) command is run. Specifying this option to a value other than off requires the [-memory_bist](#) option to be set to auto.

When set to off, the pre-DFT DRC rules [DFT_C11](#), [DFT_C12](#) and [DFT_C13](#) will not be run. This value is typically used if you intend to insert OCC in a later insertion pass, or if you do not intend to run memory BIST TCK mode.

When set to on, the pre-DFT DRC rules [DFT_C11](#), [DFT_C12](#) and [DFT_C13](#) will run and identify all asynchronous, generated and branch clocks in the direct fanin of memories and blocks for TCK injection. This value is typically used for chip design levels in which you wish to run memory BIST TCK mode by using the [tck_clock_only](#) Patterns property.

When set to [internal_sources_only](#), the pre-DFT DRC rules [DFT_C11](#), [DFT_C12](#) and [DFT_C13](#) will run and identify all generated and branch clocks in the direct fanin of memories and blocks for TCK injection. This value is typically used for [sub_block](#) and [physical_block](#) design levels in which you wish to run memory BIST TCK mode though use of the [tck_clock_only](#) Patterns property.

Clocks that already have TCK injection will not be identified for additional muxing if it is detected that it already exists in the [DFT_C11](#) analysis, even if this option is specified with the on or [internal_sources_only](#) values. Existing TCK injection points will only be identified if they are controlled by the [tck_select](#) DFT signal.

Note

 This option should not be used if you intend to add OCC’s to your design. The OCC’s will implement the TCK muxing if the [ijtag_host_interface](#) property in the [OCC](#) wrapper is not set to “none”. Additionally for this usage, the OCC TCK muxing must be described in ICL by specifying the [include_clocks_in_icl_model](#) property to “on” if you wish to use the [tck_clock_only](#) Patterns property.

Examples

The following example specifies that memory test is to be considered in the current DFT insertion pass. Notice that specifying memory test is equivalent to setting the other three related options to auto.

```
set_dft_specification_requirements -memory_test on
```

is equivalent to

```
set_dft_specification_requirements -memory_bist auto \
                                  -memory_bisr_chains auto \
                                  -memory_bisr_controller auto
```

The following example only sets memory BIST to auto which means that even if the current design contains repairable memories, memory BISR chains will not be inserted. You can use the “[set_memory_instance_options -use_in_memory_bisr_dft_specification](#)” command to control this on a per memory basis.

```
set_dft_specification_requirements -memory_bist auto
```

Related Topics

[get_dft_specification_requirement](#)

[set_diagnosis_options](#)

Context: patterns -scan_diagnosis

Mode: setup, analysis

Specifies the type of diagnosis performed by the diagnose_failures command.

Usage

```
set_diagnosis_options
[-Mode Auto | Scan | Chain | Iddq ]
[-Verify_patterns ON [Noverbose | Verbose] | OFF]
[-Report Default | percentage]
[-AT_speed ON | OFF]
[-MAX_suspects Default | limit]
[-Time_limit OFF | seconds]
[-WALL_time_limit OFF | seconds]
[-Pattern_sampling {failing_pattern_limit | OFF} {passing_pattern_limit | OFF}]
[-FAILurefile_mismatch_verbosity Default | number | All]
[-BRidge_analysis ON | OFF]
[-CHEck_last_shift_value ON | OFF]
[-X2B_mismatch Ignore Warning [Noverbose | Verbose] | Error]
[-B2X_mismatch Ignore Warning [Noverbose | Verbose] | Error]
[-CELL_internal_analysis ON | OFF]
[-MAX_Faulty_chains failed_chain_threshold]
[-Chain_diagnosis_result Cell | CHain]
[-CHAIN_LIB_internal_pathname OFF | ON]
[-ABORT_DIAGNOSE_COMPOUND_faults ON | OFF]
[-ABORT_DIAGNOSE_MANY_faulty_chains ON | OFF]
[-ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability number]
[-ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability number]
[-IGNORE_TOOL_version OFF | ON]
[-INCLude_fail_signatures_size maximum_rows_per_table | MAX]
[-INCLUDE_BRIDGE_to_power ON | OFF]
[-GRoss_delay ON | OFF]
[-COMPOUND_Hold_time_fault_diagnosis ON | OFF]
[-INCLUDE_DFM_rules ON | OFF]
[-INCLUDE_RCD_constants ON | OFF]
[-EXPected_value ON | OFF]
[-internal_failing_cells ON | OFF]
[-cycle_offset offset]
[-dynamic_partition master]
[-missing_rcd_action Ignore | Warn | Error ]
[-cell_port_bridge_analysis auto | on | off] [-cell_faults udfm_file]
[-landmark_polygon_limit integer]
[-include_core_instance_name ON | OFF]
```

Description

Specifies the type of diagnosis performed by the diagnose_failures command.

The set_diagnosis_options command specifies global settings that determine the behavior of subsequent [diagnose_failures](#) commands. These global settings remain in effect until another set_diagnosis_options command is issued.

The set_diagnosis_options command is optional, as there are invocation defaults for all the settings. If no arguments are specified, a list of the current settings displays.

Arguments

- **-Mode** [**Auto**](#) | [**Scan**](#) | [**Chain**](#) | [**Iddq**](#)

A switch and literal pair that determines what type of diagnosis is performed. The following options are available:

Auto — A literal that enables Tesson Diagnosis to determine what type of diagnosis to run based on the contents of the failure file. If the failure file contains a *chain_test* keyword and chain test failures, a chain diagnosis is performed. If the failure file contains scan test failures, a scan diagnosis is performed.

If the failures include a padding cycle of a scan chain, a chain diagnosis is run.

If chain failures are not preceded by a *chain_test* keyword, use the -faulty_chain switch with the [diagnose_failures](#) command to run chain diagnosis.

If the failure file contains both chain failures and scan failures, you must use the *scan_test* and *chain_test* keywords to identify the data. This is the default.

In Auto mode, the tool automatically detects IDDQ failure data from the fail log and runs IDDQ diagnosis.

Scan — A literal that enables scan diagnosis only. If the failure file contains scan test failures, a scan diagnosis is performed.

If a padding cycle of a scan chain fails, the scan chain is defective, and the entire chain is masked out before running logic diagnosis. The diagnosis resolution may suffer due to masked chain(s). You should run chain diagnosis in this situation.

Chain — A literal that enables chain diagnosis only. If the failure file contains a *chain_test* keyword and chain test failures, a chain diagnosis is performed otherwise the failure file is skipped.

Iddq — A literal that enables IDDQ diagnosis only.

- **-Verify_patterns** [**ON**](#)[[**Noverbose**](#) | [**Verbose**](#)] | [**OFF**](#)

A switch and literal pair that enables or disables verification of the external patterns specified with the [diagnose_failures](#) command. Test pattern verification is critical to accurate diagnosis results, so you should not disable it unless patterns have been previously verified.

ON — A literal that enables verification of the external patterns. If verification is enabled, the first [diagnose_failures](#) command entered simulates the external patterns

and compares the capture values with the expected values in the external patterns. When a mismatch occurs, verification stops and an error is issued. Use the [report_failures](#) -Pdet command to determine the complete set of failing patterns.

This verification is performed only once per session for a particular set of patterns.

ON Noverbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool does not print out the cell pathname associated with the mismatched bit. When you use this switch, the ‘read failure’ does not produce the cell paths either.

ON Verbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool prints out the cell pathname associated with the mismatched bit. This is the default.

OFF — A literal that disables verification of the external patterns.

- -Report **Default** | *percentage*

A switch and value pair that changes the diagnosis report generated by the tool. You have the following two report choices:

Default — A literal that specifies to report the following for each symptom:

- All suspects with score greater than or equal to 80.
- Up to top three suspects even if the suspects’ scores are less than 80.
- If the suspect beyond the top three has the same score with the 3rd suspect, the tool will report the suspect only if:
 - i. The suspect has same score as suspect #3, and the suspect can explain same or higher number of failing patterns.
 - a. In the case the suspect has same score and fail_match, the suspect will be reported only if it has the same passing pattern mismatch.

Note

 For definitions of the terms “symptom” and “suspect”, refer to the [Tessent Glossary](#).

percentage — An integer in the range 1 to 100 that specifies a percentage of the suspects the tool should report for each symptom. If you specify 75, for example, and there are four suspects for a particular symptom, the tool will report the top 75% (top three) of them.

- -AT_speed **ON** | **OFF**

A switch and literal pair that specifies performing at-speed failure diagnosis. See “[At-Speed Failure Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for complete information, including the conditions Tessent Diagnosis uses when diagnosing at-speed failures.

- **-MAX_suspects Default | limit**

A switch and value pair limiting the number of suspects per symptom in the diagnostics report. Choose one of the following options:

Default — A literal that specifies reporting 100 suspects per symptom.

limit — An integer that specifies to report this number of suspects per symptom.

- **-Time_limit Off | seconds**

A switch and value pair that specifies a time limit, in seconds, for diagnosing a single failure file. This time includes the failure file verification time, but not the pattern verification time. By default, the test patterns and each failure file is verified for consistency when a diagnosis is run.

If the specified time limit is reached before completion of the diagnosis, the diagnosis is aborted. A “Diagnosis aborted due to time limit (<time>)” message displays and no suspects are reported.

- **-WALL_time_limit Off | seconds**

A switch and value pair that specifies a wall time limit, in seconds, for diagnosing a single failure file. When you specify a wall time limit (**seconds**) and a diagnosis job exceeds the limit, then the tool aborts the diagnosis run with an error message.

Note

 The wall time limit is independent of the CPU time limit (using the **-Time_limit** switch). If you set to both the CPU and wall time limit, then the tool aborts the diagnosis when either limit is reached.

If the wall time limit is reached during diagnosis, the tool produces an incomplete diagnosis report file. If you specify the [diagnose_failures](#) command with the **-csv** switch, then the tool also creates an empty CSV file.

- **-Pattern_sampling {*failing_pattern_limit* | Off} {*passing_pattern_limit* | Off}**

Optional switch and value pair that specifies pattern limits for logic diagnosis. Use this switch to limit the patterns used for logic diagnosis and reduce run time. By default, all failing and passing patterns are used for diagnosis. To ensure adequate diagnosis resolution, the pattern sample must be 32 or greater.

Logic diagnosis patterns with several capture cycles require more simulation time and have less resolution. To avoid losing resolution, Tessent Diagnosis selects patterns with the least amount of capture cycles for the diagnosis. Pattern sampling options include:

failing_pattern_limit — Must be an integer of 32 or higher.

passing_pattern_limit — Must be an integer of 32 or higher. Specified number of passing patterns are selected from the first passing patterns in the set.

- **-FAILurefile_mismatch_verbosity Default | number | All**

A switch and value pair that controls the number of pattern mismatches displayed during the verification of the failure file. You have the following report choices:

Default — A literal that specifies to display 20 failing patterns.

This is the default.

number — An integer that specifies to report this number of failing patterns.

All — A literal that specifies to report all failing patterns.

- **-BRidge_analysis ON | OFF**

A switch and literal pair that determines if diagnosis includes an analysis to identify bridge suspects. A bridge suspect is identified when faulty behavior is observed for the implicated nets and the appropriate aggressor-victim conditions are met. Currently 2-way and 3-way bridges are considered and this switch is applicable to both.

Options include:

ON — Bridge analysis is enabled. Default setting.

OFF — Bridge analysis is disabled.

There are two cases where it may be desirable to disable bridge analysis.

- **Case 1** — Because the analysis is currently based only on logical responses, it is possible that a bridge suspect may not be physically possible. In cases where bridge suspects are verified as not physically possible, you may want to rerun the diagnosis with bridge analysis turned off.
 - **Case 2** — In cases where very few failing patterns are captured for a failing device, and bridge analysis may return a large number of suspects, you can turn off the bridge analysis to reduce the number of suspects.
- **-CHEck_last_shift_value ON | OFF**

An optional switch and literal pair that controls whether last shift values are considered for transition launching in resistive open diagnosis as well as cell internal diagnosis.

ON — Enables last shift values consideration for transition launching in resistive open diagnosis. This is the default.

OFF — Disables considering last shift values for transition launching in resistive open diagnosis. Use this option if you believe that the shift is very slow speed such that the transition launched from last shift is irrelevant.

- **-X2B_mismatch Ignore Warning [Noverbose | Verbose] | Error**

A switch and literal pair that determines how X2B mismatches are handled during diagnosis. X2B mismatches occur during pattern verification when the simulator returns binary values instead of the expected X values.

The tool most likely introduces X2B mismatches because it removes cell constraints when performing pattern verification and diagnosis. This allows the tool to perform diagnosis using one flat model for potentially multiple modes, each of which have different

constraints, or to use flat models that were written at different times. You can ignore these X2B mismatches.

Options include:

Ignore — This is the default. Does not display X2B mismatch information. Pattern verification will pass.

Warning Noverbose — Warns at the end of pattern verification of any X2B mismatches. Pattern verification can still pass.

Warning Verbose — Prints during pattern verification any mismatched X2B bits. Pattern verification can still pass.

Error — Prints the X2B mismatches for the first mismatched pattern and pattern verification will fail, which terminates the diagnosis or verify_patterns command.

- **-B2X_mismatch** **Ignore** **Warning[Noverbose| Verbose]** | **Error**

A switch and literal pair that determines how B2X mismatches are handled during diagnosis. B2X mismatches occur during pattern verification when the simulator returns X values instead of the expected binary values. These mismatches could be introduced by different problems, for example, software enhancements to the simulator or using a mask file when reading patterns.

Options include:

Ignore — Does not display B2X mismatch information. Pattern verification will pass.

Warning Noverbose — This is the default. Warns at the end of pattern verification of any B2X mismatches. Pattern verification can still pass.

Warning Verbose — Prints during pattern verification any mismatched B2X bits. Pattern verification can still pass.

Error — Prints the B2X mismatches for the first mismatched pattern and pattern verification will fail, which terminates the diagnosis or verify_patterns command.

- **-CEL1_internal_analysis** **ON** | **OFF**

An optional switch and literal pair that determines if the tool performs analysis to diagnose defective library cells and distinguish them from defects on nets interconnecting library cells.

- **-MAX_Faulty_chains** *failed_chain_threshold*

An optional switch and integer pair that specifies the maximum number of failing scan chains within a single datalog that are diagnosed using chain fault model. When a failure log has multiple failed_chains, there can either be multiple defects in scan chains, or a single defect affecting the scan control signals. The default *failed_chain_threshold* is 2.

- If there are one or two failed_chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis. This is the default behavior of the tool.
- If failed_chains is greater than two, then the tool assumes the defect is on a global control signal. If “**set_diagnosis_options** -

“ABORT_DIAGNOSE_MANY_faulty_chains” is OFF, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tool aborts diagnosis for this case.

Changing *failed_chain_threshold* from the default value (2) causes the following effects:

- If *failed_chains* is greater than *failed_chain_threshold*, then the tool assumes the defect is on a global control signal. If “set_diagnosis_options -ABORT_DIAGNOSE_MANY_faulty_chains” is OFF, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tools aborts diagnosis for this case.
- If there are one, two, ... *failed_chain_threshold* faulty chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis on each of the defects.

For example, a failure file contains six failed_chains. You can diagnose each of these six failed-chains as individual defects by setting this switch as follows:

```
set_diagnosis_options -max_faulty_chains 6
```

Note

 The run time is roughly linearly increased with the total number of failed_chains.

- -Chain_diagnosis_result **CELL** | **CHain**

Optional switch and literal pair that determines if a preliminary high-level chain diagnosis is performed versus a standard diagnosis. A preliminary high-level chain diagnosis only reports one failure per chain; the dominant failure.

Options include:

CELL — Runs the standard diagnosis of chain plus scan patterns and chain plus scan failures to determine cell defects. Default setting.

CHain — Runs a preliminary high-level diagnosis of the chain failures in the failure file and reports the name, length, and fault type of the chains that failed.

- -CHAIN_LIB_internal_pathname **OFF** | **ON**

An optional switch and literal pair that reports the internal instance pathname within the library for each scan latch in a suspect scan cell.

When set to ON, the tool reports this information in the chain diagnosis report’s “lib_internal_pathname” column. If there is no instance name, the tool reports “”. If you specify the **diagnose_failures** command with the -csv switch, then the tool also includes this information in the CSV output.

See “[Chain Diagnosis Section](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

- **-ABORT_DIAGNOSE_COMPOUND_faults ON | OFF**

An optional switch and literal pair that specifies whether or not the tool diagnosis the logic part of compound defects during chain diagnosis. The default to “ON”, meaning the tool does not diagnose the logic portion of compound defects.

See “[Compound Diagnosis Abort Logic Diagnosis Part](#)” in the *Tessent Diagnosis User’s Manual* for a complete discussion.

- **-ABORT_DIAGNOSE_MANY_faulty_chains ON | OFF**

An optional switch and literal pair that instructs the tool to skip diagnosing when #faulty_chains is larger than the *failed_chain_threshold* specified with the set_diagnosis_options -MAX_Faulty_chains switch. The default is “ON”.

When you set this option to OFF and the number of faulty chains it greater than specified, the tool does not report each faulty chain.

See “[Abort Conditions for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for a complete discussion.

- **-ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability *number***

An optional switch and integer pair that specifies the minimum chain fail probability for the tool to abort chain diagnosis. The default is 1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

See “[Abort Conditions for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for a complete discussion.

- **-ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability *number***

An optional switch and floating point pair that specifies the minimum scan pattern fail probability for the tool to abort chain diagnosis. The default is 0.1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

See “[Abort Conditions for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for a complete discussion.

- **-IGNORE_TOOL_version OFF | ON**

An optional switch and literal pair that instructs the tool to ignore the tool version when loading a startup cache. If the tool versions are different—that is, the startup cache was created by an older version—the tool issues a warning. By default, this option functions as follows with the following verify_patterns options:

- With -create_startup_cache: Has no impact. The tool creates the same startup cache whether -ignore_tool_version is on or off.
- With -update_startup_cache: Causes a conflict error.
- With -load_startup_cache: Disables the tool version check and uses the startup cache regardless of the tool version used to generate or update the cache.

When set to OFF, the tool issues an error and does not load the startup cache if the tool version used to create the startup cache is older than the current tool version.

- **-INCLude_fail_signatures_size *maximum_rows_per_table* | MAX**

An optional switch and integer pair that specifies the maximum number of rows per table for reporting failure signature information in the diagnosis report. The default is 256. If you do not want the failure signatures included in the diagnosis report, then specify 0 (zero) with this switch. Conversely, if you require all signature information, then use the keyword MAX with this switch. See “[Failure Signature Information in the Diagnosis Report](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

- **-INCLUDE_BRIDGE_to_power ON | OFF**

An optional switch and literal pair that specifies performing analysis and reporting of potential bridging defects between the STUCK suspect net, and a power or ground line during layout-aware diagnosis. The default is “ON”.

See “[Power and Ground Bridge Reporting](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

- **-GRoss_delay ON | OFF**

An optional switch and literal pair that specifies performing analysis and reporting of gross delay defects. The default is “OFF”.

See “[Gross Delay Defect Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

- **-COMPOUND_Hold_time_fault_diagnosis ON | OFF**

An optional switch and literal pair that specifies performing analysis and reporting of slow clock compound hold-time defects. The default is “OFF”. This option only applies to hold times on shift paths, not capture paths.

See “[Slow Clock Compound Hold-Time Diagnosis](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

- **-INCLUDE_DFM_rules ON | OFF**

An optional switch and literal pair that specifies to perform DFM hit reporting. The default is “ON”.

See “[Diagnosis for Design for Manufacturability Analysis](#)” in the *Tessent Diagnosis User’s Manual*.

- **-INCLUDE_RCD_constants ON | OFF**

An optional switch and literal pair that specifies to populate the LDB with RCD constants. The default is “ON”.

See “[Diagnosis for Root Cause Deconvolution Analysis](#)” in the *Tessent Diagnosis User’s Manual*.

- **-EXPECTED_value ON | OFF**

An optional switch and literal pair that specifies whether to verify expected pattern values against actual pattern values and return the results in the failure log. When set to OFF, the tool suppresses this check and ignores any missing expected values.

Caution

 The -expected_value switch turns off failure log verification, which means that you will not know whether there is a discrepancy between the failure log and the patterns. Use this switch with extreme caution.

- **-internal_failing_cells ON | OFF**

An optional switch and literal pair that specifies whether to enable internal scan cell profiling. See “[Internal Scan Cell Profiling for Compressed Patterns](#)” in the *Tessent Diagnosis User’s Manual*.

- **-cycle_offset *offset***

An optional switch and integer pair that specifies to adjust the cycles in a cycle-based failure file by the value *offset*. Tessent Diagnosis uses the adjusted cycles for failure verification and diagnosis. See “[Cycle Offset Adjustment for Failure Files](#)” in the *Tessent Diagnosis User’s Manual*.

- **-dynamic_partition master**

An optional switch and literal pair that specifies to generate a startup cache for dynamic partitioning-based diagnosis. You must specify this command before you load the test patterns. It signals the tool to extract clock signatures during read_patterns.

You must use this option for dynamic partitioning-based diagnosis. Tessent Diagnosis supports startup caches generated using this option for all diagnosis uses.

- **-missing_rcd_action Ignore | Warn | Error**

An optional switch and literal pair that checks whether the LDB is populated with RCD constants. The tool checks the LDB at the beginning of the diagnosis run and performs one of the following actions when the RCD constants for the current flat file and pattern set are not present:

Ignore — Takes no action and diagnosis proceeds.

Warn — Issues a warning and diagnosis proceeds. This is the default.

Error — Issues an error and diagnosis halts.

Use this switch to ensure that the LDB contains the RCD constants required for RCD analysis with Tessent YieldInsight. The warning and error outputs remind you to run create_feature_statistics, as needed, to generate the RCD constants.

- **-cell_port_bridge_analysis auto | on | off**

An optional switch and literal pair that specifies whether to perform cell port bridge diagnosis. By default, the tool does not perform cell port bridge diagnosis for layout-aware

diagnosis but does perform cell bridge diagnosis for cell-aware diagnosis. For more information, refer to “[Cell Bridge Port Diagnosis Reporting](#).”

- **-cell_faults *udfm_file***

An optional switch and string pair that, when specified, enables cell-aware diagnosis. Specify only one UDFM with Diagnosis View file. Refer to “[Running Cell-Aware Diagnosis](#).”

- **-landmark_polygon_limit *integer***

An optional switch and integer that controls the number of landmark polygons that are written. The tool uses a proximity-to-defect algorithm to limit the number of landmark polygons for global signals such as power, ground, and scan enable. Specify an integer greater than 99. The default is 10000.

- **-include_core_instance_name **ON** | **OFF****

An optional switch and literal pair that specifies whether to generate the layout-aware hierarchical diagnosis report with chip-level pin and net names rather than the default core-level names. For more information, refer to “[Running Layout-Aware Diagnosis Using a Core-Level LDB](#)”.

Examples

Example 1

The following example loads patterns from two files into the external pattern database for the tool. The failure file, *scan_failure_file*, contains just a *scan_test* section, so the tool only performs a scan diagnosis for that file.

```
read_patterns pat_file1
read_patterns pat_file2 -append
set_diagnosis_options -mode scan
diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv
```

Example 2

The following example runs a preliminary diagnosis on *my_failure_file*.

```
read_patterns pat_file1
set_diagnosis_options -chain_diagnosis_result chain
diagnose_failures my_failure_file
```

Example 3

The following example shows a preliminary diagnosis report.

```
tracking_info_begin
tracking_info_end
#faulty_chains=2 #symptoms=0 #suspects=0 CPU_time=0.00sec
fail_log=fail_logs/my_failure_file
Chain chain1 has error, fault type is STUCK_AT_0, the length of this chain
is 102.
Chain chain3 has error, fault type is STUCK_AT_1, the length of this chain
is 101.
```

Related Topics

[diagnose_failures](#)

[report_failures](#)

[write_failures](#)

set_display

Context: unspecified, all contexts

Mode: all modes

Sets the DISPLAY environment variable from the tool's command line.

Usage

`set_display display_name`

Description

Sets the DISPLAY environment variable from the tool's command line.

The set_display command sets the DISPLAY environment variable to display_name without exiting the currently running application. If you invoke the tool in command line mode (the default), then the DISPLAY variable is not required in order to use most commands successfully. If the variable is not set, however, and you issue a command such as open_visualizer that requires a GUI, the tool will not be able to show the applicable window. This is because the display connection provided by the variable is absent. The set_display command enables you to set the variable, if needed for the display, without the inconvenience of exiting to the shell in order to set it.

Note

-  This command affects the DISPLAY setting within the currently running application only.
When you exit the tool, the setting in the invocation shell will be what it was when you invoked the tool.
-

Arguments

- ***display_name***

A required string that specifies a valid display setting for the machine on which the tool is running.

Examples

The following example sets the DISPLAY variable when “open_visualizer” fails to open DFTVisualizer. The example also uses the system command to pass a Linux “echo \$DISPLAY” command to the shell in order to check the variable’s setting.

```
open_visualizer
//=====
// Warning: The DFTVisualizer window(s) will not be accessible
// for this session due to the lack of a valid X display
// connection.
//
// Try using the set_display command to set your DISPLAY
// variable. OR start/restart your Xserver.
//=====
```

```
system echo $DISPLAY  
set_display my_workstation:0.0  
system echo $DISPLAY  
my_workstation:0.0
```

Related Topics

[system](#)

set_dofile_abort

Context: unspecified, all contexts

Mode: all modes

Specifies whether the tool aborts or continues dofile execution if it detects an error condition.

Usage

`set_dofile_abort ON | OFF | exit`

Description

Specifies whether the tool aborts or continues dofile execution if it detects an error condition.

By default, if an error occurs during the execution of a dofile, processing stops, and the line number causing the error in the dofile is reported. The `set_dofile_abort` command lets you turn this functionality off so that the tool continues to process all commands in the dofile. In this case, the tool aborts the current Tcl construct (if/for/while) but continues to process the remaining commands in the dofile. You can use the Tcl `catch` command or the Tesseract `catch_output` command to manage errors and prevent the tool from aborting the current construct.

Arguments

- **ON**

A literal that halts the execution of a dofile upon the detection of an error. When in batch mode the tool exits on error, but when in interactive mode the tool returns to the session prompt. This is the default upon invocation of the tool.

- **OFF**

A literal that forces dofile processing to complete commands in a dofile regardless of error detection. If an error is encountered in a Tcl if, for, or while construct within the dofile file, the tool exits the construct immediately but continues execution of the next sequential command outside the construct. Note, this option does not affect the Tcl `source` command which completely stops execution of the source file when an error is encountered.

- **exit**

A literal that directs the tool to exit the session if it detects an error while executing a dofile regardless of whether invoked in batch or interactive mode.

Examples

Example 1

The following example sets the `set_dofile_abort` command off to ensure that all commands in `test1.dofile` are executed.

```
set_dofile_abort off
dofile test1.dofile
```

Example 2

In the following example dofile, if cmd1 returns an error and set_dofile_abort is set to On, cmd2 does not execute and the tool exits the dofile.

If cmd1 returns an error and set_dofile_abort is set to Off, cmd2 still does not execute but the tool processes the remainder of the dofile. You can use the Tcl catch command or the tesseract catch_output command around cmd1 to manage the errors generated by cmd1 and allow the tool to continue with cmd2 even if cmd1 generated an error. The set_dofile_abort command only affects how the error is handled when it propagates up the nested command stack and reaches the dofile level.

```
if {1} {
    cmd1
    cmd2
}
if {1} {
    cmd3
    cmd4
}
```

Related Topics

[catch_output](#)

[dofile](#)

set_drc_handling

Context: unspecified, all contexts

Mode: all modes

Specifies how the tool handles design rule violations.

Usage

```
set_drc_handling {drc_id} [Error | Warning | NOTe | Ignore]
  [NOVerbose | Verbose]
  [NOAtpg_analysis | Atpg_analysis] // C{1,3,4,5,6}, D9, E{4,5,8,10,11,12}
  [ATPGC] // E10
  [-Mode A clk_name] // C3, C4
  [-Interval number] // C3, C4
  [-CONSERvative {ON | OFF}] // C6
  [-IGNORE_SYNChronous_clock_groups {ON | OFF}] // C6
  [-SKip_procedure {TEst_setup | OFF}] // E4
  [-TIMing_exception_check {ON | OFF | Auto}] // E5
  [-CONSIDer_primary_outputs_observable {ON | OFF}] // E5
  [-MASk_interactions {ON| OFF }] // C22
  [-INteractions {edge | any)} // C22
  [-Mode {Sequential | Combinational}] // E10
  [-SIMULATE_ACROSS_Internal_primary_inputs {ON | OFF}]
  [-auto_fix]
  [-INITialization_cycle {ON | OFF}]
  [-ALLOW_COMMon_scan_cells_in_multiple_chains {ON | OFF}]
  [-SCan_chain_tracing {Aggressive | Conservative})
```

Description

Specifies how the tool handles design rule violations.

The set_drc_handling command specifies the handling of the messages for the following design rule checks:

- Scan cell RAM rules checking (A rules)
- Some BIST rules checking (B2 and B4 rules)
- Clock rules checking (C rules)
- Certain Pre-DFT Clock Rules (DFT_C Rules)
- Custom DRC rules you may have registered using the [register_drc](#) command
- Data rules checking (D rules)
- Extra rules checking (E rules)
- EDT Finder rules checking (F rules)

- EDT rules checking (K rules)
- Some Procedure rules checking (P rules: P29-P33, P86, P89, P90, P92, and P94)
- Some Scan Pattern Retargeting rules checking (R rules)
- Trace rules checking (T rules)
- Some Power-aware rules checking (V rules)
- Certain Timing rules checking (W rules)

You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses either the handling from the last set_drc_handling command or, if you did not change the handling, the Design Rules Checker's invocation default.

Note

 The set_drc_handling command does not support the scannability (S) rules except for the S3 and S5 rules. Unlike the other DRC rules, setting the handling type to "Error" does not turn on the transcript verbosity for the S3 rule. The verbosity can only be turned on via the optional literal "verbose".

The set_drc_handling command does not support any Flattening design rules (FN, FP, and FG rules). Use the [set_flattener_rule_handling](#) command to specify how design rule violations are handled for the Flattening rules.

The set_drc_handling command does not support the K15 rule.

Each rules violation has an associated occurrence message and summary message. The tool only displays the occurrence message for either error conditions or if you specify the Verbose option for that rule. The tool also displays the rule identification number in all rules violation messages.

The Atpg_analysis option provides test generation analysis when performing rules checking for some clock (C) rules, for some data (D) rules, and for some extra (E) rules. For example, if Atpg_analysis is enabled for clock rule C1, and the tool simulates a clock input as X, the rule violation occurs when it is possible for the test generator to create a test pattern while that clock input is on, all defined clocks are off and all constrained pins are at their constrained state.

Note

 When Atpg_analysis is enabled, the tool requires some additional CPU time and memory to perform the test generation analysis. (The Atpg_analysis option is enabled by default for rules C1, E10, E11 and E13; you can disable it for these rules by specifying the Noatpg_analysis option.)

Arguments

- ***drc_id***

A required non-repeatable literal that specifies the identification of the exact design rule violation whose message handling you want to change.

For more information on the design rules, refer to “[Design Rule Checking](#)”.

- Error

An optional literal that specifies for the tool to both display the error occurrence message and immediately terminate the rules checking.

- Warning

An optional literal that specifies for the tool to display the warning summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rules violation.

- NOTe

An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation

- Ignore

An optional literal that specifies for the tool not to display any message for the rule’s violations. The tool must still check some rules and they must pass to allow later performance of certain functions.

- **NOverbose**

An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display a summary message indicating the total number of occurrences of the rule violation. This is the default.

If DRC handling is set to Error or Ignore, the tool will behave as follows regardless of whether you specify the Noverbose (or Verbose) option:

- Error—The tool will stop at each error and display detailed instance-specific violation information to help you debug it.
- Ignore—The tool will not perform the check and therefore, the consequences of the check on ATPG, whether good or bad, will not occur. Also, the tool will not display or store instance-specific violation information.

- Verbose

An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display the occurrence message for each occurrence of the rule violation. All instance-specific violations (maybe hundreds or thousands) will be listed, one message per instance violating the rule. To obtain a summary message instead, use the Noverbose option.

If the DRC's handling is set to Error or Ignore, the tool will behave as follows regardless of whether you specify the Verbose (or Noverbose) option:

- Error—The tool will stop at each error and display detailed instance-specific violation information to help you debug it.
- Ignore—The tool will not perform the check and therefore, the consequences of the check on ATPG, whether good or bad, will not occur. Also, the tool will not display or store instance-specific violation information.

- **NOatpg_analysis**

An optional literal that specifies for the tool not to use test generation analysis when performing rules checking. This is the default except for rules C1, E4, E10, E11 and E13.

- **Atpg_analysis**

An optional literal that specifies for the tool to use test generation analysis when performing rules checking for clock rules (such as C1, C3, C4, C5 and C6), some D rules (such as D6 and D9) and some E rules (such as E4, E5, E8, E10, E11 and E12). This is the default for rules C1, E4, E10, E11 and E13.

For clock rules C3 and C4, the Atpg_analysis option generates a check of the clocks of the source and sink to see if they are gated off.

Note

 To use the constraint values during the D6 rule analysis, you must use the Atpg_analysis option.

- **ATPGC**

An optional literal that specifies for the design rules checker to use all the current ATPG constraints when analyzing E10 rule violations. You can also use the “[add_atpg_constraints -Static](#)” command to do the same thing.

- **-Mode A *clk_name***

A switch, a literal (A), and a string triplet that specifies the name of a clock on which you want the tool to perform further analysis to screen out false C3 and C4 clock rules violations. The tool performs this extra analysis on one clock only.

For more information on using the -Mode option, refer to the [C3](#) and [C4](#) descriptions.

- **-Interval *number***

An optional switch and integer pair that you can only use with C3 and C4 clock violations to specify how often you want the tool to display a message during the ATPG analysis of those violations. The *number* argument indicates multiples of violation occurrences that cause the tool to display a message. The default is 0.

The message includes the number of sequential elements that the tool checked, the number of sequential elements remaining to check, the current number of ATPG passes during the

C3 or C4 clock rules checking, and the current CPU time used by the tool for clock rules checking.

The value of the *number* parameter must be either zero or a positive integer. You can only specify one *number* value that the tool uses for both the C3 and C4 violations. If you issue multiple set_drc_handling commands (one for C3 and one for C4) that specify different values for the *number* argument, the tool uses the last interval value you specified.

- **-CONSERvative {ON | OFF}**

An optional switch and literal for [C6](#) DRC analysis. The default is OFF.

The default C6 checking may miss some C6 violations when [set_clock_restriction](#) OFF is used. A more conservative analysis can be enabled by setting the -CONSERvative to ON and will identify those C6 violations. However, it is generally recommended not to turn off clock restriction such that the -CONSERvative switch does not usually need to be used.

- **-IGNORE_SYNChronous_clock_groups {ON | OFF}**

An optional switch and literal for the [C6](#) DRC analysis. The default is OFF.

By default, C6 DRC analysis treats clocks in the same synchronous clock group as if they are equivalent. You can use the “set_drc_handling C6 -ignore_synchronous_clock_groups on” to turn off this behavior.

For more information on synchronous clock groups, see [add_synchronous_clock_group](#).

- **-SKip_procedure {TEst_setup| OFF}**

An optional switch and literal pair that affects the tool's handling of the E4 rule. The default option specifies to skip these checks when simulating the test_setup procedure. The tool will continue to check for contention during other test procedures. The Off option specifies for the tool to perform E4 rule checks for the test_setup procedure, as well as other procedures, during DRC.

It is very common during test_setup, when a design is being initialized, to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention, and the tool may issue many E4 violation messages. If you wish to be notified about this type of bus contention during test_setup, use the Off option to include test_setup for E4 rule checks. This switch has no effect if you have previously set E4 handling to Ignore.

- **-TIMing_exception_check { ON | OFF | Auto}**

An optional switch and literal that enables the timing exception check for the E5 DRC. The default is Auto, in which case the tool will consider timing exception paths during E5 DRC (checking for X sources) when the tool is in the patterns -scan context and will make the handling of E5 DRC an error when the tool is in “Logic_bist” context. When the setting is Auto and the tool is not in “Logic_bist” context, the tool will ignore timing exception paths for E5 DRC.

- **-CONSIDer_primary_outputs_observable { ON | OFF }**

An optional switch and literal that enables checking the propagation of X sources to a primary output for the E5 DRC. The default is off, in which case the tool does not check whether X sources reach a primary output during E5 DRC.

- **-MASK_interactions { ON | OFF }**

An optional switch and literal pair that enables or disables the addition of false paths to avoid simulation mismatches due to C22 DRC violations. The default on, enables the tool to add the cross domain false paths. The tool generates these messages as part of DRC to summarize the C22 false path addition:

```
// 2 cross domain false paths are added to prevent simulation
mismatches due to C22 DRC violations.
// Use "set_drc_handling C22 -mask_interactions off" in SETUP mode
to disable automatic addition of false paths.
```

- **-INTERactions { edge | any }**

An optional switch and literal pair that specifies the interaction type for C22 cross clock path checks. When the default, edge, is used, the tool will only check same edge interactions for C22. Use “-interactions any” to check cross clock path checks with any interactions.

- **-Mode {Combinational | Sequential}**

An optional switch and literal for the E10 rule.

Combinational — Default upon invocation. This option performs bus contention mutual-exclusivity checking. This checking differs from rule E4 in that it does not check for this condition during test procedures.

Sequential — Considers the inputs to a single level of sequential cells behaving as “staging” latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enables of the tri-state devices except through the sequential cells.

- **-SIMULATE_ACROSS_Internal_primary_inputs { ON | OFF }**

An optional switch and literal pair that changes how the tool simulates internal PI's during DRC. Internal PIs are primary inputs you previously defined using the [add_primary_inputs](#) or [add_clocks](#) commands.

After you set this switch to ON in setup mode, during test procedure simulation the tool starts off simulating an internal PI as an internal signal by carrying over its driver's simulation value until the point when you first supply a simulation value by explicitly forcing this PI in any of the following test procedures: test_setup, load_unload, or shift.

This switch is needed for cases where you may have a cut point (such as for capture modeling), yet you need test_setup to be simulated normally as if the cut point were not present. This is commonly needed when using IJTAG within test_setup, where you want the IJTAG network simulated normally even if the network includes a cut point. For more

information about using IJTAG and test procedures, refer to “[Test Setup and Test End Procedures](#)” in the *Tessent IJTAG User’s Manual*.

- **-auto_fix {ON | OFF}**

An optional switch used to enable or disable auto fixing for specific DRC rules. Currently only [DFT_C9](#) supports this feature and the auto fix is ON by default. Use this switch to disable auto corrections for DFT_C9 as follows:

```
set_drc_handling dft_c9 -auto_fix off
```

If you are registering your own DRC rules using the [register_drc](#) command and you have used the -allow_auto_fix switch, it will be possible to disable the auto_fix using the set_drc_handling command.

- **-INITialization_cycle {ON | OFF}**

An optional switch and literal pair that controls whether the tool forces all clocks to their off states and all constrained pins to their constrained values prior to simulation of the test_setup procedure during DRC. By default, the tool performs these forces, so you do not have to explicitly specify the forces, which matches the default initialization cycle the tool includes in saved patterns. If you have occasion to write patterns without this initialization cycle ([write_patterns](#) -Noinitilization), issue a set_drc_handling -Initialization Off command and rerun DRC before saving patterns.

Tip

 If DRC is run with initialization enabled, turning off initialization when saving patterns ([write_patterns](#) -Noinitilization) may result in simulation mismatches during pattern verification in a timing based simulator. The converse (DRC run with initialization disabled and patterns saved with initialization enabled) is typically not a problem.

The argument choices are as follows:

ON — A literal that specifies to initialize the clocks and constrained pins to their off states and constrained values, respectively, prior to DRC simulation of the test_setup procedure. This is the invocation default.

OFF — A literal that specifies not to initialize the clocks and constrained pins prior to DRC simulation of the test_setup procedure.

- **-ALLOW_COMMON_SCAN_CELLS_IN_MULTIPLE_CHAINS {ON | OFF}**

An optional switch and literal pair that specify whether or not to allow scan cells to be shared among scan chains. Note that there are some restrictions on scan cell sharing, which result in a T2 violation when not observed:

- Scan cell sharing is not allowed for compressed chains driven by a decompressor.
- The shared scan cells must be at the logical scan cell boundary. A library model with multiple scan cells cannot be partially shared. In other words, you cannot have the merged point within the state elements of a logical scan cell.

-
- **-SCan_chain_tracing{Aggressive | Conservative}**

An optional switch and literal pair that specify whether or not scan chain tracing passes with a T25 violation. The choices are as follows:

Aggressive — Scan chain tracing passes with a T25 violation. This is the default. For more information on T25 violations, see the “[T25](#)” section.

Conservative — Scan chain tracing does not pass with a T25 violation. In this case, scan chain tracing fails, design rule checking stops, and the tool issues a T3 violation. For information on correcting T3 violations, see the “[T3](#)” section.

Examples

The following example specifies rule checking E4 to be an error:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
set_drc_handling e4 error
set_system_mode analysis
```

Related Topics

[report_drc_rules](#)

[get_drc_handling](#)

[set_driver_restriction](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool allows multiple drivers on buses and multiple active ports on gates.

Usage

`set_driver_restriction Off | ON`

Description

Specifies whether the tool allows multiple drivers on buses and multiple active ports on gates.

The `set_driver_restriction` command lets you specify for the tool to report a contention problem whenever there are multiple nets driving values onto a bus, or when multiple ports are active on an individual gate. The default upon tool invocation is to allow multiple driving nets on a bus or multiple active ports on a gate as long as the signals are driving the same value. If multiple signals are on and are not driving the same values, then the tool flags it as contention.

However some design processes only allow a single driver to be on at a time regardless of whether the signals are driving the same values. The `set_driver_restriction` command lets you place this same restriction on the tool.

Arguments

- **Off**

A literal that specifies to allow multiple drivers to be on for a bus and multiple ports to be active for a gate as long as the driving signals are of the same value, and therefore there is no contention. This is the default behavior when you invoke the tool.

- **ON**

A literal that restricts buses to only have one driver on at a time and gates (dff and latches with multiple clocks) to only have one active port; the tool flags multiple active drivers or ports as contention problems.

Examples

The following example creates a strict contention checking environment. The first command specifies for the tool to check for contention on both multiple port gates and buses. If there is a contention problem where signals are driving different values, the tool reports an error and stops the simulation. The second command further restricts the contention checking environment by not allowing multiple drivers to even drive the same values.

```
set_contention_check on -error -all
set_driver_restriction on
```

Related Topics

[set_contention_check](#)

set_edt_abort_analysis_options

Context: dft -edt, patterns -scan (EDT On)

Mode: all modes

Specifies the number of EDT Aborted (EAB) test cubes on which to perform EAB fault analysis.

Usage

```
set_edt_abort_analysis_options -max_cubes_to_analyze {unlimited | maxCubes}
```

Description

Specifies the number of EAB test cubes on which to perform EAB fault analysis.

The `set_edt_abort_analysis_options` command is used to set the maximum number of allowed EAB test cubes to be stored during the EDT session. By default, up to 1000 test cubes are stored. If you specify the unlimited keyword, then the tool stores all EAB test cubes generated from the EDT session.

You must issue this command before using the [create_patterns](#) command.

See “[EDT Aborted Fault Analysis](#)” in the *Tessent TestKompress User’s Manual* for a complete discussion.

Arguments

- **-max_cubes_to_analyze**{unlimited | *maxCubes*}

A required switch and value pair that specifies the number of EAB test cubes. Choose one from the following:

unlimited — A literal that specifies all EAB test cubes for analysis.

maxCubes — An integer that specifies the number of EAB test cubes for analysis. The default is 1000.

Related Topics

[report_edt_abort_analysis](#)

set_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Controls whether the tool automatically identifies EDT logic and updates scan chain pins during test pattern generation.

Usage

```
set_edt_finder [OFF | ON] [-Verbose {OFF | ON}]
```

Description

Controls whether the tool automatically identifies EDT logic and updates scan chain pins during test pattern generation. This capability is referred to as EDT Finder. By default, EDT Finder is “on”.

When EDT Finder is on, the F DRCs are enabled, in addition to the normal K DRCs. The F DRCs provide more precise identification of EDT logic errors. Additionally, when EDT Finder is on, EDT DRC run time is greatly reduced.

This command can be used in conjunction with Core Mapping for ATPG. For more information, see “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan and ATPG User’s Manual*.

You can use this command to get updated scan pin information when your design hierarchy changes after the EDT logic is generated. For more information, see “[Updating Scan Pins for Test Pattern Generation](#)” in the *Tessent TestKompress User’s Manual*.

EDT Finder uses the defined scan chain pins if they match what EDT Finder itself found in the logic. That is, if in shift mode EDT Finder determines that there is a controlling path between the defined scan chain pin and the pin that EDT Finder found, EDT Finder will use the defined scan chain pin.

If no arguments are specified, the current EDT Finder settings display.

Note

 EDT Finder is automatically disabled in cases that are not supported—for example, in BIST mode. Because EDT Finder is enabled by default, DI faults are automatically identified in EDT logic. You can disable this behavior by executing the following command:

```
set_fault_subclass_analysis -disable DI.EDT
```

For more information, see “[Fault Sub-classes](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **OFF**
An optional literal that disables EDT Finder.
- **ON**
An optional literal that enables EDT Finder. This is the default.
- **-Verbose {OFF|ON}**
An optional switch and literal pair that enables the transcription of informational messages about the EDT logic found. The transcription of informational messages is disabled upon invocation of the tool.

Examples

The following example shows the output of EDT Finder for a design with two channels and eight internal scan chains. EDT Finder output is highlighted in bold.

```
dofile results/generated_edt.dofile
add_scan_groups grp1 results/generated_edt.testproc
add_scan_chains -internal chain1 grp1 /top_i/si1 /top_i/so1

add_scan_chains -internal chain2 grp1 /top_i/si2 /top_i/so2
set_edt_pins update EdtUpdate
set_edt_pins input_channel 1 ChannelInput_1
set_edt_pins output_channel 1 ChannelOutput_1
set_edt_pins input_channel 2 ChannelInput_2
set_edt_pins output_channel 2 ChannelOutput_2
set_mask_decoder_connection -mode_bit 1 5
...
set_system_mode analysis

// Flattening process completed, cell instances=221, gates=471, PIs=16,
// POs=11, CPU time=0.00 sec.
// -----
// Begin circuit learning analyses.
// -----
//
// -----
// Begin EDT finder analyses.
// -----
// EDT Finder completed, EDT blocks=1, scan chains=8, CPU time=0.02 sec.
// -----
// Begin scan chain identification process, memory elements = 54.
// -----
...
```

Related Topics

[report_edt_finder](#)

set_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Specifies the location in which the tool places the EDT logic.

Usage

```
set_edt_instances [-Edt_logic_top edt_instance_name_or_pathname]
                  [-Compactor compactor_instance_name_or_pathname]
                  [-Decompressor decompressor_instance_name_or_pathname]
                  [-Block_location path_to_top_of_block]
```

Description

IP Creation phase:

Specifies the location in which the tool places the EDT logic.

In the IP Creation phase, you use this command with the -Block_location switch to specify the parent instance in which the EDT logic is to be inserted. See Example 1. Note, if you use any switch except -Block_location in this phase, the tool ignores the switch and issues a warning.

By default, the EDT logic is instantiated at the top level of the design.

Pattern Generation phase:

Specifies an instance name/location for existing EDT logic or for the design block that contains the EDT logic (EDT block).

You can use this command in the Pattern Generation phase to:

- Tell the tool the new name of an already renamed EDT logic block, decompressor, or compressor. See Example 3.
- Name each EDT block uniquely, using the -Block_location switch, when an EDT block is used multiple times. See Example 4.

The set_edt_instances command operates only on the current EDT block which is the last block specified by either the [add_edt_blocks](#) or [set_current_edt_block](#) commands. For more information, see “[Modular Compressed ATPG](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- **-Edt_logic_top *edt_instance_name_or_pathname***

A required switch and string pair that specifies an instance name or pathname for the design block containing the EDT logic (EDT block). Names specified with this switch must be

unique. If the specified `edt_instance_name_or.pathname` is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- **-Compactor** `compactor_instance_name_or.pathname`

A required switch and string pair that specifies an instance name or pathname for the compactor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified `compactor_instance_name_or.pathname` is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- **-Decompressor** `decompressor_instance_name_or.pathname`

A required switch and string pair that specifies an instance name or pathname for the decompressor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified `decompressor_instance_name_or.pathname` is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- **-Block_location** `path_to_top_of_block`

A required switch and string pair that specifies the location of the current EDT block within the top-level netlist.

During EDT IP Creation, use this switch to specify the location in which the tool places the EDT logic.

During Pattern Generation, use this switch to identify each EDT block uniquely when an EDT block is used multiple times, by specifying its parent instance pathname.

Examples

Example 1

The following command is issued during EDT logic creation to insert the EDT logic in the instance, `pad_east/test_blk`, instead of at the top level of the design:

```
set_edt_instances -block_location pad_east/test_blk
```

Example 2

The following example specifies the name of the instance that contains the EDT logic:

```
set_edt_instances -edt_logic_top my_core_edt_i
// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
```

Example 3

The following example specifies the instance names of the decompressor and compactor for the EDT logic and reports the current instance locations:

```
set_edt_instances -decompressor my_core_edt_decompressor_i \
                  -compactor my_core_edt_compactor_i
// Found compactor at path:
//      "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
//      "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

report_edt_instances

```
// edt_logic_top
//   Setting: my_core_edt_i
//   Path    : /my_core_edt_top/my_core_edt_i
//
// decompressor
//   Setting: my_core_edt_decompressor_i
//   Path    : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//   Setting: my_core_edt_compactor_i
//   Path    : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Example 4

The following example specifies unique names for instances of the same EDT block in a top-level modular netlist:

```
set_edt_mapping on
add_edt_blocks my_core1_a

// my_core1_a is the current EDT block.

set_edt_instances -block_location /my_core1_a_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

// edt_logic_top
//   Setting: my_core1_edt_i
//   Path    : /my_core1_a_edt_i/my_core1_edt_i
...

add_edt_blocks my_core1_b

// my_core1_b is the current EDT block.

set_edt_instances -block_location /my_core1_b_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

// edt_logic_top
//   Setting: my_core1_edt_i
//   Path    : /my_core1_b_edt_i/my_core1_edt_i
...
```

The set_edt_mapping command enables the requisite mapping of block-level dofile commands to the top-level.

Related Topics

[add_edt_blocks](#)
[report_edt_instances](#)
[set_current_edt_block](#)

[**set_edt_mapping**](#)

[set_edt_mapping](#)

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.

Usage

```
set_edt_mapping {Off | ON} [-Verbose {Off | ON}]
```

Description

Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.

Note

 This command and the EDT Mapping functionality have been superseded by the newer Core Mapping for ATPG feature. Please use the newer capability described in “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan and ATPG User’s Manual*.

The set_edt_mapping command determines whether or not the tool performs the automatic mapping necessary to reuse, for top-level pattern generation, the dofiles the tool generated during block-level IP creation. You must set this command to “on” prior to running block-level dofiles to set up for top-level pattern creation.

For more information about the modular Tessent TestKompress flow or the reuse of block-level dofiles, refer to the “[Modular Compressed ATPG](#)” chapter of the *Tessent TestKompress User’s Manual*.

Note

 Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_edt_mapping commands until you specify a different setting

Arguments

- **Off**

A literal that turns off automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase. This is the default upon invocation of the tool.

- **ON**

A literal that enables the automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase.

- -Verbose {**OFF** | **ON**}

An optional switch and literal pair that enables or disables the tool's transcription of informational messages whenever it changes any of the following block-level dofile objects as a result of automatic mapping:

scan group name
 scan chain name
 pin pathname
 pin name

The transcription of informational messages is disabled upon invocation of the tool.

Examples

Example 1

The following example enables the automatic mapping of commands from the block-level dofiles to the top-level design hierarchy, then executes the commands in one of the block-level dofiles:

```
set_edt_mapping on
delete_edt_blocks -all
// All blocks deleted. No EDT block now selected as current block.

add_edt_blocks created_core1
// created_core1 is the current EDT block.

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile
// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Will use top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// command: add_clocks 0 SCLK
// command: add_clocks 0 CCLK
...
```

Example 2

The following example shows the transcription that would have occurred had the -Verbose switch been in effect for the commands of the preceding example:

```
set_edt_mapping on -verbose on
delete_edt_blocks -all
```

```
// All blocks deleted. No EDT block now selected as current block.

add_edt_block created_core1
// created_core1 is the current EDT block.

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile

// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Will use top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
// Note: Block level chain name (chain1) automatically translated to
// created_core1_chain1.
// Note: Block level path (/m8_i/scan_in1) automatically translated to
// /x1/m8_i/scan_in1.
// Note: Block level path (/m8_i/scan_out1) automatically translated to
// /x1/m8_i/scan_out1.
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// Note: Block level chain name (chain4) automatically translated to
// created_core1_chain4.
// Note: Block level path (/m8_i/scan_in4) automatically translated to
// /x1/m8_i/scan_in4.
// Note: Block level path (/m8_i/scan_out4) automatically translated to
// /x1/m8_i/scan_out4.
// command: add_clocks 0 SCLK
// Note: Block level "add_clocks" command for SCLK will be verified
// during DRC.
// command: add_clocks 0 CCLK
// Note: Block level "add_clocks" command for CCLK will be verified
// during DRC.
```

Related Topics

[add_edt_blocks](#)
[delete_edt_blocks](#)
[set_edt_instances](#)

set_edt_options

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -edt, dft -scan, and patterns -scan contexts only)

Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.

Usage

```
set_edt_options [OFF | ON | Auto]
  [-CHannels integer]
  [-INPut_CHannels integer]
  [-Used_input_channels integer]
  [-OUtput_CHannels integer]
  [-LOCATION {External | Internal}]
  [-BYPASS_Chains integer]
  [-OVERShift_cycles integer]
  [-SIngle_bypass_chain {ON | OFF}]
  [-BYpass_logic {ON | OFF | Use_existing_bypass_chains}]
  [-Clocking {Edge | Level}]
  [-PIpipeline_logic_levels_in_compactor {OFF| integer}]
  [-LOCKup_cells {ON | OFF}]
  [-LONGest_chain_range {min_number_cells max_number_cells}]
  [-RESET_signal {OFF| Asynchronous}] [-COMpactor_type {XPress | BAsic}]
  [-PULSE_EDT_Before_shift_clocks {ON | OFF}]
  [-FOrce_1hot_masking {OFF | ON}]
  [-RETIme_chain_boundaries {ON | OFF}]
  [-SEParate_control_data_channels ON | OFF]
```

For dft -scan

```
set_edt_options [-Location {External | Internal}]
  [-Channels integer]
```

For Hybrid TK/LBIST Flow IP Creation

```
set_edt_options -lbist_misr_input_ratio { integer | auto }
  [-reset_signal asynchronous]
  [-SIngle_bypass_chain {ON | OFF}]
  [-BYpass_logic {ON | OFF}]
  [-chain_mask_register_ratio ratio]
```

For Hybrid TK/LBIST Flow Fault Simulation

```
set_edt_options [-decompressor_seed hex_string]
```

Description

Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.

To create EDT logic for a basic application, you only need to specify the number of channels.

When set to Off, the tool can be used for ATPG and behaves the same as Tesson FastScan in command-line mode.

Use the [report_environment](#) command to display the current settings.

Settings are retained over multiple `set_edt_options` commands until a different setting is specified.

Hybrid TK/LBIST Flow Specifics

Sets options for EDT IP creation and LogicBIST fault simulation.

IP Creation: This command specifies options specific to the EDT/LogicBist blocks.

Fault Simulation: This command describes the initial seed of the PRPG.

Refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- **OFF**
An optional literal that disables compression (EDT) mode and enables analysis mode.
- **ON**
An optional literal that enables compression (EDT) mode.
- **Auto**
An optional literal that enables automatic detection of EDT mode (compression). Note, this option is only available in the patterns -scan context.
- **-CHannels *integer***
An optional switch and integer pair that specifies the number of external scan channels. Use this switch when the number of input channels is the same as the number of output channels.
Use this switch to set up the number of scan channels the EDT logic is configured with. During pattern generation, this switch is used to specify the number of scan channels the EDT logic contains. The dofile created when the EDT logic is generated, contains a `set_edt_options` command that specifies the number of channels for test pattern generation.
- **-INPut_CHannels *integer***
An optional switch and integer pair that specifies the number of scan input channels. Use this switch only if the number of input channels differs from the number of output channels.
- **-Used_input_channels *integer***
An optional switch and integer pair that specifies the number of EDT IP input channels used for pattern generation. This option is not valid during IP creation. This switch can only be used in setup mode.

You can configure the EDT IP to use a subset of the input channels. For example, you might generate the EDT IP to support some number of input channels, but then when the IP or core is embedded into the design, only a subset of the channels can be driven due to scan port limitations.

Note

 It is important to know that, during EDT IP creation, the tool considers all channels as possible inputs/outputs for the bypass chains. If you plan to tie any of the input channels at some point, you need to consider constraining the used input channels prior to EDT IP creation.

Use of this option can provide a flexible test compression planning flow. Since a core could be designed independent of the rest of the SoC, it could be difficult to determine the optimal number of scan input channels during the core design phase. You may pick a conservative compression ratio by using enough scan input channels at the IP creation phase. When integrating all cores into the SoC, if using only a subset of the input channels is sufficient for a core, you can use this feature to specify the used input channels and apply a specific constant value on the unused input channels on the boundary of the EDT IP during shift.

The following usage conditions apply:

- To use this switch, you must set the following in the IP Creation Phase:

```
set_edt_options -separate_control_data_channels ON
```

The only exception is if you are using the basic compactor and no low-power controller.

- Only data channels with highest channel indexes can become unused. For example, an EDT block has one control channel (channel1) and three data channels (channel2, channel3, channel4). If you want to only use 3 input channels, you can only tie channel4 to “0”. If you want to only use 2 input channels, you can only tie both channel 3 and channel4 to “0”. You must have at least one data channel.
- If the unused input channels are connected to top-level primary inputs, patterns at these pins must hold at “0”s. This can be accomplished by constraining the unused input channels using the add_input_constraints command:

```
add_input_constraints XXX -C0
```

or

```
add_input_constraints XXX -C1 (if there is an inversion
between the tied pin and the EDT channel input)
```

Otherwise, the tool issues DRC violations during system mode transition.

- If the unused input channels are driven by internal signals, these channels must be controlled in the design to ensure that “0”s are injected into the decompressor from these unused channels.

- If the unused input channels are driven by logic that is setup to provide constant values during shift (for example, a TDR that is loaded with “0” in test_setup), then you need to apply the appropriate setup patterns.

Refer to “[Core Instance Parameters](#)” in the *Tessent TestKompress User’s Manual*.

- **-OUtput_CHannels *integer***

An optional switch and integer pair that specifies the number of scan output channels. Use this switch only if the number of output channels differs from the number of input channels.

- **-BYPASS_Chains *integer***

An optional switch and integer pair that determines how many scan chains are created for bypass mode. The specified value must be equal to or less than the number of bypass chains created by default. By default, the number of bypass chains created equals the number of input/output channels. If the number of input and output channels differ, the smaller number is used.

For dual compression configuration applications, this switch must be issued after both configurations are defined.

- **-OVERShift_cycles *integer***

An optional switch and integer pair that adds a specified number of shift cycles to a test pattern set. Use this switch to balance out core test pattern sets when integrating them at the chip level in the SoC flow.

This switch can be used before saving out a particular pattern set to adjust the number of shift cycles. This is typically useful for mapping core level patterns to the chip level in the SoC pattern mapping flow. During core level pattern generation, the patterns can be saved as is. While integrating core level patterns at the chip level, the differences in the number of shift cycles for the different pattern sets corresponding to the different cores will be known. The switch can then be used to add the additional cycles to the pattern set with lesser number of cycles to balance out all the pattern sets. This facilitates pattern merging at the chip level.

- **-SIngle_bypass_chain {ON | OFF}**

An optional switch and literal pair that creates compression logic with a dual bypass chain configuration. Use this option to create a bypass configuration that concatenates all scan chains together into one bypass chain in addition to the default bypass configuration. For more information, see “[Dual Bypass Configurations](#)” in the *Tessent TestKompress User’s Manual*.

Hybrid TK/LBIST Flow Only: -SIngle_bypass_chain {ON | OFF} The default is ON.

Turn this switch off to turn off generating single bypass chain logic on a per-block basis.

Caution

 Turning this switch off is not recommended because doing so also turns off LBIST pattern diagnosis.

- **-LOCAtion {External | Internal}**

An optional switch and literal pair that specifies a location for the EDT logic relative to the netlist read in. Options include:

External—Creates a new top level “wrapper” module and instantiates both the design netlist read in and the EDT logic within it. The logic is thus kept external to the design netlist. This is the default.

Internal—Instantiates the EDT logic in the existing top level of the design netlist.

- **-BYpass_logic{ON | OFF | Use_existing_bypass_chains}**

An optional switch and literal pair that specifies a location for the bypass logic. Bypass logic allows you to bypass the compression circuitry and run traditional ATPG with a tool such as Tesson FastScan. Bypass logic is transparent during normal operation. Options include:

ON—Includes bypass logic in the EDT logic. This is the default.

OFF—The tool does not add any logic; it assumes that a bypass does not exist and inserts no logic.

Use_existing_bypass_chains—Uses bypass logic already in the core logic. In this case, the tool inserts two muxes for each EDT channel (not chain) in order to select between the compressed and uncompressed path. [Example 3](#) demonstrates how to use this option.

Hybrid TK/LBIST Flow Only: Turn this switch off to turn off generating bypass logic. When you do so, the tool also automatically turns off generating single bypass chain logic.

Caution

 Turning this switch off or specifying Use_existing_bypass_chains is not recommended because doing so turns off LBIST pattern diagnosis.

- **-CLocking {Edge | Level}**

An optional switch and literal pair that determines what type of architecture logic is used for the EDT logic. Options include:

Edge—Specifies DFF-based logic. You can use DFF-based logic for designs that are LSSD-based, mux-DFF-based, or that incorporate a mixture of these two architectures. This is the invocation default.

Level—Specifies latch-based logic. This option is for pure LSSD designs only.

- **-PIpipeline_logic_levels_in_compactor {OFF | integer}**

An optional switch and literal pair that enables pipeline stages in the compactor of the EDT logic. Options include:

OFF—Disables pipeline stages in the EDT logic. This is the default.

integer—Includes pipeline stages in the EDT logic compactor(s), and specifies the maximum number of logic levels (2-input XOR gates) allowed in the compactors before inserting a pipeline stage.

- -LOCKup_cells {**ON** | **OFF**}

An optional switch and literal pair that determines whether lockup cells are inserted in the EDT logic.

Caution

 You should leave this switch set to *on*. Disabling lockup cell synthesis can lead to DRC violations in the pattern generation phase and inability to generate valid patterns. For more information, see “[Understanding Lockup Cells](#)” in the *Tessent TestKompress User’s Manual*.

Options include:

ON — Includes lockup cells in the EDT logic as needed between the decompressor and the core, between the core and the compactors (if pipelining is used), and between scan chains when they are concatenated for bypass mode. This is the default setting.

OFF — Disables lockup cells in the EDT logic.

- -LONGest_chain_range *min_number_cells max_number_cells*

An optional switch and integer pair that defines a range for the length of the longest scan chain. Use this switch to avoid having to regenerate the logic for minor changes in the length of the chains after the logic is created.

Note

 This switch should be used when creating EDT logic with the skeleton flow. For more information, see “[Longest Scan Chain Range Estimate](#)” in the *Tessent TestKompress User’s Manual*.

Options include:

min_number_cells — Specifies the minimum number of cells in the longest scan chain. When you use the -longest_chain_range switch with separate control and data channels, the *min_number_cells* impacts the number of control channels that are created, according to this formula:

$$\text{Number of EDT control channels} = \text{Ceil}(\text{total number of control bits} / \text{min_number_cells})$$

This can result in higher pattern count if the final design has more control channels than necessary.

For example: If you specify the minimum longest chain length as 50, and the design has 100 control bits, the generated hardware will have two control channels. However, if the actual longest chain length in the final design is 100 or more, one control channel is sufficient, but the EDT IP wastes one control channel and has one fewer data channel, which increases the pattern count unnecessarily. You should estimate the *min_number_cells* as close as possible to its final design.

max_number_cells — Specifies the maximum number of cells in the longest scan chain. A too large *max_number_cells* value may increase the decompressor size a little.

- -RESet_signal {**OFF** | **Asynchronous**}

An optional switch and literal pair that specifies whether an asynchronous reset signal, `edt_reset`, is included in the EDT logic. In most cases such a signal is not needed, but it can be used if design requirements dictate that all the sequential elements in the design are resettable. Options include:

OFF—Disables the asynchronous reset. This is the default.

Asynchronous—Includes an asynchronous reset signal for all sequential elements in the EDT logic and a new top-level pin named “`edt_reset`” to control it. You can rename this pin or share it with a functional pin in the design core using the [set_edt_pins](#) command.

Note

 You should define the reset as a clock primary input using the [add_clocks](#) command during logic creation. This accounts for the signal off state (high or low) when writing out the EDT logic. If the signal is undefined, an active high is assumed at the boundary of the EDT logic. If the signal is shared with a global reset that is active low, you must manually insert an inverter for the reset signal before it arrives at the EDT logic boundary.

- -COMpactor_type {**XPress** | **BASIC**}

Optional switch and literal pair that determines which compactor is used in the EDT logic. Options include:

XPress—Xpress compactor. The Xpress compactor increases compression, especially for designs that generate unknown (X) values. Default setting.

BASIC—Basic compactor. The basic compactor is the first generation compactor.

For more information, see “[Understanding Compactor Options](#)” in the *Tessent TestKompress User’s Manual*.

- -PULse_EDt_Before_shift_clocks {**ON** | **OFF**}

Optional switch and literal pair that determines whether the EDT logic clock is pulsed before the scan chain clock. By default, the EDT logic and scan chain clock are pulsed simultaneously. For more information, see the “[Pulse EDT Clock Before Scan Shift Clocks](#)” section in the *Tessent TestKompress User’s Manual*.

- -FOrce_1hot_masking {**OFF** | **ON**}

Optional switch and literal pair that determines whether Tessent TestKompress generates test patterns that observe a single scan chain or multiple scan chains on each channel of the Xpress compactor. Options include:

OFF—Depending on unknown states and targeted faults, Tessent TestKompress determines which scan chains to observe. Default behavior.

ON—One scan chain is observed on each compactor channel. This setting should be used for debug purposes. Generating test patterns with this setting increases test pattern count.

This switch is only valid when the Xpress compactor is used. Observing only one scan chain output per channel allows you to easily map a failure on a channel to a specific scan cell. For more information on the Xpress compactor, see “[Understanding Compactor Options](#)” in the *Tessent TestKompress User’s Manual*.

- **-SEParate_control_data_channels on| off**

An optional switch that when enabled (set to on) specifies that the tool automatically determines the number of control channels and inserts control bits behind the control channels only. This is needed for channel sharing between non-identical blocks. For more information, see “[EDT IP Creation With Separate Control and Data Input Channels](#)” *Tessent TestKompress User’s Manual*.

You apply this setting per EDT block. For a design with multiple EDT blocks, you can enable some of the blocks using “-separate_control_data_channels on” and disable the remaining blocks using “-separate_control_data_channels off”.

Note

 You can only use this option if the user-specified number of input channels is greater than or equal to one plus the number of control channels; otherwise, the tool issues an error and aborts.

- **-RETIme_chain_boundaries ON | Off**

An optional switch that specifies whether to add retiming cells (lockup cells) at scan chain inputs and outputs. These cells are added to ensure that each scan chain starts with an LE register and ends with a TE register. This switch replaces bypass lockup cells with retiming lockup cells that are part of the EDT scan chains; this eliminates the need for bypass-only lockup cells. For more information on bypass cells as part of the EDT logic, see “[Differences Based on Inclusion/Exclusion of Bypass Lockup Cells in EDT Chains](#)” in the *Tessent Shell Reference Manual*.

Retiming cells are added only when lockup cells are inserted with the “-lockup_cells ON” option. These cells are included as part of both EDT and EDT-bypass scan chains.

ON — Specifies to include retiming cells in EDT chains. This is the default for the Hybrid TK/LBIST flow (dft -edt -logic_bist context).

Off — Specifies to *not* include retiming cells in EDT chains. This is the default for TestKompress EDT IP creation (dft -edt context).

For complete information on including retiming cells in EDT chains, see section “[Lockup Cell Analysis For Bypass Lockup Cells Included as Part of the EDT Chain](#)” in the *Tessent TestKompress User’s Manual*.

- **-lbist_misr_input_ratio { integer | auto } (Hybrid TK/LBIST Flow IP Creation Only)**

A required switch that specifies the ratio of the number of scan chains over number of MISR inputs. The specified value should be a power of 2.

If the value is 1, then all scan chain outputs are connected to the MISR. This is the default value. Setting this property to auto reverts the input ratio to the default 1 value.

If the value is greater than or equal to #scan_chains/#channels, then all EDT channel outputs are connected to the MISR.

If any value in between is used, then taps are taken from the middle of EDT compactor XOR tree corresponding to the value specified.

This argument only specifies the number of MISR inputs; the size of the MISR is implicitly determined based on this input. The minimum size of the MISR is at least 24 bits.

If this switch is not specified, a default MISR of 24 bits is used. When the design has more than 24 output channels, the default MISR size is increased to be greater than or equal to the number of output channels.

- **-chain_mask_register_ratio *ratio***

An optional switch and integer pair that specifies the number of scan chains controlled by a single EDT chain mask register bit. By default, the ratio is 1:1; there are as many bits in the chain mask register as there are number of scan chains.

You can use the -chain_mask_register_ratio switch to minimize the area impact of the chain mask register. For example, specifying 3 for *ratio* means a 3:1 ratio with each chain mask register bit controlling 3 scan chains. Specify the ratio on a per EDT block basis.

The larger the ratio you specify, the more you trade off resolution for area impact. When any scan chain connected to a chain mask register bit is masked, all the other scan chains connected to the bit are also masked automatically.

Refer to “[Scan Chain Masking](#)” in the *Hybrid TK/LBIST User’s Manual* for more information.

- **-decompressor_seed *hex_string* (Hybrid TK/LBIST Flow Fault Simulation Only)**

An optional switch and hexadecimal pair that specifies the initial seed of the PRPG. This seed applies to both the warm-up patterns and the actual patterns.

The LogicBIST fault simulation dofile includes this switch with the hardware default value for arguments.

If you intend to perform fault simulation with a different starting PRPG seed, then modify the *hex_string* argument to specify the desired seed.

Examples

Example 1

The following example sets up a EDT logic for a design with four scan chains. The EDT logic is set up for one scan channel and no bypass circuitry.

```
add_scan_groups grp1 atpg.proc
add_scan_chains chain1 grp1 edt_si1 edt_so1
add_scan_chains chain2 grp1 edt_si2 edt_so2
add_scan_chains chain3 grp1 edt_si3 edt_so3
add_scan_chains chain4 grp1 edt_si4 edt_so4
add_clocks 0 clk
set_edt_options -channels 1 -bypass_logic off
set_system_modeanalysis
```

Example 2

The following example generates EDT logic with two input channels and one output channel. The number of bypass chains defaults to 1, corresponding to the smaller of the two specified channel values.

```
set_edt_options -input_channels 2 -output_channels 1
```

Example 3

The following example demonstrates the use of the use_existing_bypass_chains option to the -bypass_logic switch. In this example, you have two bypass chains since you plan to insert EDT with two input and two output channels. In this example, your my_bypass signal is active low.

1. Tell the tool about the bypass chains:

```
set_edt_options -bypass_logic use_existing_bypass_chains -bypass_chains 2
```

2. Tell the tool about the input and output of each chain:

```
set_bypass_chains -bypass_chain_number 1 -pins scan_in1 scan_out1 \
-edt_chains chain1 chain2 chain3 chain4
```

```
set_bypass_chains -bypass_chain_number 2 -pins scan_in2 scan_out2 \
-edt_chains chain5 chain6 chain7 chain8
```

The tool inserts a mux on scan_in1 and scan_in2 to select between the EDT channels and the bypass path. The tool inserts a mux on the output so that you will have two muxes for each EDT channel.

The tool will control the newly inserted bypass muxes using the default edt_bypass pin which will be added unless you use the set_edt_pins command to tell the tool about your bypass control. If you do use the set_edt_pins command as shown here, the tool will use my_bypass instead of the default edt_bypass pin.

```
set_edt_pins bypass my_bypass
```

The tool assumes that the pin you specify here (my_bypass) is active high. If your bypass enable pin is active low, you must edit the generated RTL for EDT and add an inversion on the my_bypass signal coming into the EDT logic. With this change, when you set your bypass pin low, it will configure all of your bypass muxes and the tool-inserted muxes to be in bypass mode.

Example 4 (Hybrid TK/LBIST Flow IP Creation)

This command during IP creation is used to connect the scan chains to MISR by combining four scan chain outputs using XOR gates to each MISR input.

```
set_edt_options -lbist_misr_input_ratio 4
```

Example 5 (Hybrid TK/LBIST Flow IP Creation)

For a design with 20 scan chains, specify the following command during IP creation to group 4 scan chains per chain mask register bit:

```
set_edt_options -chain_mask_register_ratio 4
```

The resulting chain mask register will be a 5-bit register instead of a 20-bit register (the default).

Example 6 (Hybrid TK/LBIST Flow Fault Simulation)

The following example shows how to modify the PRPG seed in analysis mode during fault simulation:

```
set_edt_options -decompressor 00000001  
set_system_mode analysis  
simulate_patterns -source bist -store_patterns all  
set_edt_options -decompressor 00110101  
simulate_patterns -source bist -store_patterns all
```

Related Topics

[report_edt_configurations](#)
[report_edt_pins](#)
[report_scan_volume](#)
[set_bypass_chains](#)
[set_compactor_connections](#)
[set_edt_pins](#)
[write_edt_files](#)

[set_edt_pins](#)

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.

Usage

Specifying a new or shared EDT pin

```
set_edt_pins {{Input_channel | Output_channel} {index [pin_name] [internal_node_name]} |
  {edt_pin [pin_name] [internal_node_name]} |
  {pin_name -ON_edt_module}}
```

Renaming and/or inverting EDT pins

```
set_edt_pins {{Input_channel | Output_channel} {{index [pin_name]} | {edt_pin}} |
  [-Noinv | -Inv]}
```

Inserting channel pipeline stages

```
set_edt_pins {{Input_channel | Output_channel} {[index]{-Pipeline_stages #stages}}}
```

Specifying the change edge for compactor outputs

```
set_edt_pins [Output_channel -CHange_edge_at_compactor_output
  {Any | LEading_edge_of_edt_clock
  | TRailing_edge_of_edt_clock}]
  | [Output_channel index -CHange_edge_at_compactor_output
  {Any | TRailing_edge_of_edt_clock}]
```

Specifying internally driven EDT control pins

```
set_edt_pins {edt_pin} {-} [internal_node_name]}
```

Description

By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.

Except for added channel pipeline stages, a dofile generated during the EDT logic creation phase contains the correct set_edt_options commands for the pattern generation phase. If you add channel pipeline stages, you must specify those changes in the pattern generation phase with another set_edt_pins command.

Use the set_edt_pins command for the following operations:

- **Specify a new or shared EDT pin**

When an EDT channel output pin is shared with a functional output pin, a multiplexer is inserted in the top-level wrapper to share the signals between the output pins. The multiplexer is controlled by the `scan_enable` signal. If you rename an EDT pin with a name already used by a functional pin in the core design, the EDT pin shares the functional pin.

- **Rename an EDT control pin**

To rename an EDT pin, specify the `edt_pin` along with a `pin_name` argument that specifies the new name. To rename an EDT channel, specify an input/output channel and index number for the channel along with a `pin_name` argument that specifies the new name. To apply the new name to the pin at the EDT module level in the RTL, use the `pin_name -On_edt_module` switch.

- **Specify an inverted signal between the chip pin and the EDT logic**

I/O pads added outside the EDT logic sometimes invert their signal. Use the `-Inv` switch to specify the inversion of the EDT control pins during EDT logic creation. This ensures that generated test procedures and dofiles operate correctly in the pattern generation phase after the I/O pads are synthesized. The `pin_name` argument is optional if you are only changing the inversion status of an EDT pin.

- **Specify that an EDT control pin is driven internally**

For more information, see “[Internally Driven EDT Pins](#)” in the *Tessent TestKompress User’s Manual*.

- **Specify channel pipeline stages**

Channel pipeline stages are pipeline stages added outside the EDT logic between a top level channel input pin/pad and the decompressor input or compactor output.

Channel pipeline stages are typically added during top-level design integration to reduce signal propagation time between the top-level pads and the channel inputs and outputs. Channel pipeline stages are not created as part of the EDT logic, but they must be specified to generate the correct test patterns.

To specify channel pipeline stages for a particular channel during the pattern generation phase, use the `index` number of the channel, the `-Pipeline_stages` switch, and the number of pipeline stages.

- **Specify the change edge for compactor outputs**

You can use this feature to specify whether compactor output data changes on the LE time or whether the tool aligns the change edge at compactor output with the TE of the EDT clock.

By default, the compactor output data changes on the TE time of the EDT clock. There are two usage scenarios as follows:

Without compactor pipelining (compactor pipeline register has not been inserted)

- If LE is specified for all channels, a pipeline stage clocked by edt_clock (LE) is added at the compactor output; the pipeline stage is only added if any of the last scan cells change at any time other than the LE of the EDT clock. Lockup cells are added if necessary either before the pipeline stage or at the output of the scan chains on a per-compactor basis.
- When TE is specified for all output channels, a pipeline register driven by the LE of the EDT clock and a lockup cell driven by the TE of the EDT clock are added to all channels. When TE is specified for a subset of output channels, only a lockup cell driven by the TE of the EDT clock is added for the channels specified as needed.

With compactor pipelining (compactor pipeline register has been inserted)

- The compactor pipeline guarantees the compactor output changes on the LE of EDT clock because the pipeline stage inside the compactor is driven by edt_clock (LE). If the number of levels of logic is less than or equal to the number of pipeline levels specified with set_edt_options, a pipeline stage is added only when necessary to ensure an LE change. In addition, lockup cells are added when needed.
- If TE is specified for a channel, an additional lockup cell is added on the output of every compactor to ensure the TE change edge.

The dofile written out during EDT logic creation reflects the pipeline stages added to meet the speed objectives or to ensure that the output of the compactor changes at the positive edge of the EDT clock.

For more information, see “[EDT Control and Channel Pins](#)” and “[Using Pipeline Stages Between Pads and Channel Inputs or Outputs](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- **Input_channel**
Optional literal that specifies an EDT input channel pin.
- **Output_channel**
Optional literal that specifies an EDT output channel pin.
- **index**
Optional integer that specifies the index number for a specified channel pin. The index must be positive and have a valid range from one up to the total number of scan channels.
- **pin_name**
Optional replaceable string that specifies a pin name. Depending on usage, the *pin_name* may reflect the name of an existing functional pin in the core design or EDT logic, an existing pin on the EDT module, or the new name for an existing pin. This is a required argument if you are using the internal_node_name argument to create a connection between this pin and an internal node.

- *internal_node_name*

Optional replaceable string that specifies the hierarchical name of an internal node to connect an EDT pin to. This argument defines the connection between an EDT pin and the internal node of an I/O pad for internally placed EDT logic.

When defining the connection, you must also include a *pin_name* argument in the same command. If only one string is specified, the tool interprets it as the *pin_name*.

- *edt_pin*

Optional literal that specifies an EDT control pin. Literal options include:

Clock — *edt_clock* pin.

Reset — *edt_reset* pin.

Scan_enable — top-level *scan_en* pin.

Bypass — *edt_bypass* pin.

Single_bypass_chain — *edt_single_bypass_chain* pin.

Update — *edt_update* pin.

COnfiguration — *edt_configuration* pin for dual configuration applications.

Master_clock — *master_clock* pin. Use in latch-based EDT logic for LSSD designs only.

Slave_clock — *slave_clock* pin. Use in latch-based EDT logic for LSSD designs only.

Low_power_shift_en — low power shift enable pin for low-power decompressor.

Note



The tool assumes that the pin you specify for the bypass enable signal is active high. If you have pre-existing bypass logic, specified with the “*set_edt_options -bypass_logic use_existing_bypass_chains*” command, and your bypass enable pin is active low, you must edit the generated RTL for EDT and add an inversion on the bypass signal coming into the EDT logic. With this change, when you set your bypass pin low, the tool will configure all of your bypass muxes and the tool-inserted muxes to be in bypass mode.

- **[-Noinv | -Inv]**

Optional switch that specifies an EDT pin signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.

You should specify *-inv* in the following situations:

- **External flow** — Inversion between the wrapper created by the tool and the chip boundary.
- **Internal flow** — Inversion between the internal connection node for the EDT pin and the chip boundary.

Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.

- **-ON_edt_module**

Optional switch that specifies a pin on the EDT module in the EDT RTL. Use this argument to apply a change to a specified pin at the EDT module level only.

- **-Pipeline_stages #stages** (Pattern Generation Phase only)

An optional switch and non-negative integer pair that specifies the number of pipeline stages on specified input or output channels. Use the channel *index* argument to apply a number of channel pipeline stages to a specific channel. If no *index* argument is used, the specified number of channel pipeline stages applies to all input or all output channels for the current EDT block.

Use this argument during the pattern generation phase to specify the number of pipeline stages inserted between top-level pins or pads and EDT channel inputs or outputs.

If your design contains channel pipeline stages, you must also make some changes in the test procedure file to accommodate them.

Note

 The tool updates the number of pipeline stages with what was traced even if you have specified a different number using the -pipeline switch. Note, if you have disabled EDT Finder, the tool will not update the number of pipeline stages.

You must generate test patterns for existing bypass scan chains from scratch. You cannot create test patterns for existing bypass chains from identical test patterns created for EDT. For more information on using bypass chains defined in the core logic, see “[Generating EDT logic When Bypass Logic is Defined in the Netlist](#)” in the *Tessent TestKompress User’s Manual*.

For more information, see “[Using Pipeline Stages Between Pads and Channel Inputs or Outputs](#)” in the *Tessent TestKompress User’s Manual*.

- [Output_channel -CHange_edge_at_compactor_output
 {Any | LEading_edge_of_edt_clock | TRailing_edge_of_edt_clock}] |
 [Output_channel *index*-CHange_edge_at_compactor_output
 {Any | TRailing_edge_of_edt_clock}]

Optional switch and literal pair that specifies which edge the compactor output data changes on. This switch does not support latch-based EDT. Options include:

Any — Specifies that data can change at either edge and no additional logic is required.

TRailing_edge_of_edt_clock — Specifies that compactor outputs need to change on the TE edge of the EDT clock. You can use the *index* option to specify a change edge for individual channels because it does not introduce an additional shift.

When all scan chain outputs change at the same time as the LE of EDT clock, or when the compactor already has pipelining, the tool will only add a lockup cell, without a pipeline, to the end of the compactor.

When the scan chain outputs do not change at the same time as the LE of EDT clock, the tool adds a pipeline and a lockup cell. Adding a pipeline for trailing_edge only happens when all channel outputs are specified with trailing edge.

LLeading_edge_of_edt_clock — Specifies that all compactor outputs need to change on the LE edge of the EDT clock. An *index* cannot be specified because it may introduce varying amounts of shift cycles across output channels. When you specify this option and all scan chain outputs already change at LE, the tool adds a lockup cell and LE compactor pipeline register.

- **-edt_pin**

Optional switch that specifies there is no top-level pin associated with an *edt_pin* that is driven internally. Use this switch during EDT logic creation and test pattern generation for the internal flow and during test pattern generation for the external flow to prevent false K5 DRC violations.

Note

 Input and output channels must always have a corresponding top-level pin.

For more information, see “[Internally Driven EDT Pins](#)” in the *Tessent TestKompress User’s Manual*.

Examples

Example 1

The following example configures the EDT logic with two scan channels, reports the names and inversion status of the EDT pins, and issues four set_edt_pins commands that:

1. Share the EDT clock pin with a functional pin, a1.
2. Rename the dedicated EDT bypass pin to my_bypass and specifies the bypass signal is inverted between the PI pin and the EDT logic.
3. Share EDT channel 1 input pin with functional pin, portain[7].
4. Change the inversion status of the edt_update pin.

A final report_edt_pins command displays the new EDT pin settings.

```
set_edt_options -channels 2
report_edt_pins
```

```
//          Pin description      Pin name      Inversion
//          -----
//          Clock                edt_clock     -
//          Update               edt_update    -
//          Bypass mode          edt_bypass   -
//          Scan channel 1 input edt_channels_in1 -
//          "        "        " output  edt_channels_out1 -
//          Scan channel 2 input edt_channels_in2 -
//          "        "        " output  edt_channels_out2 -
//
```

```
set_edt_pins clock a1
set_edt_pins bypass my_bypass -inv
set_edt_pins input_channel 1 portain[7]
set_edt_pins update -inv
report_edt_pins
```

```
//          Pin description      Pin name      Inversion
//          -----
//          Clock                a1           -
//          Update               edt_update   inv
//          Bypass mode          my_bypass   inv
//          Scan channel 1 input portain[7]  -
//          "        "        " output  edt_channels_out1 -
//          Scan channel 2 input edt_channels_in2 -
//          "        "        " output  edt_channels_out2 -
//
```

Assume the settings defined by the preceding example are specified by similar `set_edt_pins` commands during the pattern generation phase. The next example specifies there are three pipeline stages between the scan channel 1 input pin (`portain[7]`) and the corresponding top level pad, and changes the inversion status of the pin. The example then displays the updated settings. (The Pin description column is not shown in this example in order to show the reporting command's output when pipeline stages are defined.)

```
set_edt_pins input_channel 1 -inv -pipeline_stages 3
report_edt_pins
```

```
//          Pin name      Inversion      Channel pipeline stages
//          -----
//          a1           -              -
//          edt_update   inv           3
//          my_bypass   inv           0
//          portain[7]  inv           0
//          edt_channels_out  -         0
//          edt_channels_in2 -         0
//          edt_channels_out2 -         0
```

set_edt_pins

The next example specifies new pin names to be used at the “edt” module interface for the two EDT input channels, the two EDT output channels, and the EDT clock signal:

```
//Input ch: use the name, ch_in_bus, at the edt-IP RTL module interface.
set_edt_pins input_channel ch_in_bus -on_edt_module

//Output ch: use the name, ch_out_bus, at the edt-IP RTL module interface.
set_edt_pins output_channel ch_out_bus -on_edt_module

//EDT clock: use the name tk_clk_P at the edt-IP RTL module interface.
set_edt_pins clock tk_clk_P -on_edt_module
```

The next example shows a complete dofile for an external logic location flow that implements user-specified EDT pin names at the “edt” module level:

```
dofile scan_setup.dofile

set_edt_options -channels 2 -bypass off

//
//clock: design level=PSEI, edt module level=tk_clk_P
set_edt_pins clock PSEI
set_edt_pins clock tk_clk_P -on_edt_module

//
//channel inputs: design level = AI[1]/AI[2], edt module level=ch_in_bus
set_edt_pins input_channel 1 AI[1]
set_edt_pins input_channel 2 AI[2]
set_edt_pins input_channel ch_in_bus -on_edt_module

//
//channel outputs: design level=OA[1]/OA[2], edt module level=ch_out_bus
set_edt_pins output_channel 1 OA[1]
set_edt_pins output_channel 2 OA[2]
set_edt_pins output_channel ch_out_bus -on_edt_module

set_system_mode analysis
report_edt_pins

write_edt_files created -verilog -replace
```

The next example shows “edt” and “edt_top” modules (external logic location flow), with RTL changes resulting from the use of the **-On_edt_module** switch highlighted in bold. For brevity, only the relevant portions of the “edt_top” module are shown.

```

/*****************************  

** Module: my_core_edt  

**  

*****  

module my_core_edt (tk_clk_P, edt_update, ch_in_bus, ch_out_bus,  

                    edt_scan_in, edt_scan_out);  

    input          tk_clk_P;  

    input          edt_update;  

    input [1:0]    ch_in_bus;  

    output [1:0]   ch_out_bus;  

    output [15:0]  edt_scan_in;  

    input [15:0]   edt_scan_out;  

    wire edt_mask;  

    my_core_edt_decompressor my_core_edt_decompressor_i  

        (.edt_clock(tk_clk_P),  

         .edt_update(edt_update),  

         .edt_channels_in(ch_in_bus),  

         .edt_mask(edt_mask),  

         .edt_scan_in(edt_scan_in));  

    my_core_edt_compactor my_core_edt_compactor_i  

        (.edt_clock(tk_clk_P),  

         .edt_update(edt_update),  

         .edt_scan_out(edt_scan_out),  

         .edt_mask(edt_mask),  

         .edt_channels_out(ch_out_bus));  

endmodule  

/*****************************  

** Module: my_core_edt_top  

**  

*****  

module my_core_edt_top (NMOE,  

                       NMWE,  

                       DLM,  

                       ...  

                       scan_en,  

                       edt_update);  

    output NMOE;  

    output NMWE;  

    output DLM;  

    ...  

    input   scan_en;  

    input   edt_update;  

    ...  

    my_core my_core_i (.NMOE(NMOE),  

                      .NMWE(NMWE),  

                      .DLM(DLM),  

                      ...  

                      .scan_en(scan_en));  

    my_core_edt my_core_edt_i (.tk_clk_P(PSEI),  

                               .edt_update(edt_update),  

                               ...

```

```

    .ch_in_bus({AI[2], AI[1]}),
    .ch_out_bus({edt_channels_out_2,
                  edt_channels_out_1}),
    .edt_scan_in(edt_scan_in),
    .edt_scan_out(edt_scan_out));
...
endmodule

```

Example 2

The following example specifies that the output of the compactor should change at the LE of the clock.

set_edt_pins output -change_output leading

If the compactor has any outputs that change data on TE, a pipeline stage and the necessary lockup cells are added at the output of every compactor.

Example 3

The following example uses a design with 100 scan chains and the following parameters for compression:

set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 2
set_edt_pins output -change_output leading

Pipeline stages are added after the second XOR gates stage in the compactor. Accordingly, lockup cells are also added. However, no other pipeline stages are added at the output of the compactor because the compactor pipelining guarantees that the output of the compactors will change with respect to the LE of the clock.

Assume the same design and change the pipeline stages to:

set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 6
set_edt_pins output -change_output leading

Now, no compactor pipeline stages are needed due to the minimum logic levels allowed in the compactor by the user. Under such circumstances, the last scan cell of all chains in the design are analyzed and if any of the scan outputs change at a TE edge, then the tool adds a pipeline stage at the output of all the compactors along with necessary lockup cells.

Example 4

The following example shows the commands used for a design with 100 chains and 10 channel inputs/outputs with some channel pins with pipelining.

set_edt_options -channels 10
set_edt_pins output 1 -change_edge trailing
set_edt_pins output 5 -change_edge trailing

Lockup cells are added in the compactors that have pipeline stages in the corresponding channels.

Example 5

The following example sets the output of all compactors to change at LE except for one that changes on TE because of channel output pipelining.

```
set_edt_options -channels 10
set_edt_pins output -change_edge leading
set_edt_pins output 5 -change_edge trailing
```

This guarantees that all compactor outputs will change at LE except for channel 5, which would change at TE with respect to the EDT clock.

Example 6

The following example specifies the EDT update control pin is internally driven by the */top/xyz/out* node for Tessent TestKompress logic created with the internal flow.

```
set_edt_options -location internal
set_edt_pins update - /top/xyz/out
```

The following example specifies the EDT update control pin has no corresponding top-level pin during test pattern generation.

```
set_edt_pins update -
```

Example 7

The following example specifies the connections between the channel input and output pins and their internal connection points for an internal flow. Notice that the top-level pins *port_x* and *port_y* are specified because the *internal_node_name* argument is used.

```
set_edt_options -location internal
set_edt_pins input_channel 1 port_x /ip_pad_block/port_x_pad/Z
set_edt_pins output_channel 1 port_y /ip_pad_block/port_y_pad/A
```

Related Topics

[report_edt_configurations](#)
[report_edt_pins](#)
[set_edt_options](#)
[write_edt_files](#)

set_edt_power_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Generates and inserts a power controller in the compression logic to enable low-power shift.

Usage

```
set_edt_power_controller Shift {None | Enabled | Disabled}  
    [-MIN_Switching_threshold_percentage percentage_integer]  
    [ -Full_control { OFF | ON } ]
```

Description

Generates and inserts a power controller in the compression logic to enable low-power shift.

This command controls the bias of the low-power decompressor in the EDT IP creation phase. For more information, see “[Low-Power Shift and Switching Thresholds](#)” in the *Tessent TestKompress User’s Manual*.

In the IP Creation phase, you can use this command to create the power controller, to specify its hardware capability (switching threshold), and to configure the generated dofile and test procedure file.

In the Pattern Generation phase, you can only use this command to indicate that the controller is present and whether it is enabled in the test procedure file read by the tool.

The power controller may be configured in both phases as either “enabled” or “disabled”. If you are using the same configuration (enabled/disabled) during both IP creation and pattern generation, then this command is already included in the generated dofile and you do not need to make any changes to the controller configuration. However, if you want to change how it was configured during IP creation, you need to specify the opposite value (enabled/disabled).

To generate patterns with the low power controller hardware enabled, you must do the following:

1. In the test procedure file, ensure the `edt_low_power_shift_en` is forced to the correct values during the shift procedure. For more information on the `edt_low_power_shift_en` signal, see “[Power Controller Logic](#)” in the *Tessent TestKompress User’s Manual*.
2. Issue a “`set_edt_power_controller Shift Enabled`” command in Setup mode to enable the power controller before you generate test patterns.

To generate patterns with the low power controller hardware disabled, you must do the following:

1. In the test procedure file, ensure the `edt_low_power_shift_en` is forced to the correct values during the shift procedure. For more information on the `edt_low_power_shift_en` signal, see “[Power Controller Logic](#)” in the *Tessent TestKompress User’s Manual*.
2. Issue a “`set_edt_power_controller Shift Disabled`” command in Setup mode to disable the power controller before you generate test patterns.

Note

 You cannot use this command during ATPG to change the switching threshold of an already created controller; instead, use the `set_power_control` command to change the actual switching percentage during ATPG.

For more information, see “[Low-Power Test](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- **Shift**

Required literal that specifies the generated power controller should enable low-power shift.

- **None**

Required literal that disables creation of the power controller in the compression logic. This is the default setting.

- **Enabled**

Required literal that during IP creation creates the power controller hardware as part of the compression logic and writes the ATPG dofile that enables the hardware and a test procedure file that forces the `edt_low_power_shift_en` control signal to its enabled value.

During ATPG, this option tells the tool that the low power controller is enabled in the test procedure file read by the tool.

- **Disabled**

Required literal that during IP creation creates the power controller hardware as part of the compression logic and writes the ATPG dofile that disables the hardware and a test procedure file that forces the `edt_low_power_shift_en` control signal to its disabled value.

During ATPG, this option tells the tool that the low power controller is disabled in the test procedure file read by the tool.

- **-MIN_Switching_threshold_percentage *percentage_integer***

Optional switch and integer that specifies the minimum switching threshold value used to determine power controller parameters, such as input shift register size. Valid integer options are 1 to 50. If an integer greater than 50 is specified, the tool uses 50. The default setting is 15%.

The tool uses this value to calculate the size of the power controller. The power controller must be large enough to control the number of scan chains required to achieve the minimum switching threshold specified. For more information, see “[Low-Power Shift and Switching Thresholds](#)” in the *Tessent TestKompress User’s Manual*.

- **-Full_control {OFF|ON}**

Optional switch and literal that adds one bit per scan chain to control whether that chain loads a constant value or not. This allows you to have complete control over the scan chain input masking. Choose one of the following options:

OFF — Literal that disables adding these bits to the scan chain. This is the default.

ON — Literal that enables adding one bit per scan chain.

Normally, you would only use this when your design has very few chains such that:

- The low-power register count is not significant.
- You need to control power to low thresholds or fine granularity even for those low-chain-count designs.

When you set this switch to ON, any value specified with the

-MIN_Switching_threshold_percentage switch is ignored.

Examples

Example 1

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold, and creates the dofile and test procedure file that enable the hardware during pattern generation.

```
set_edt_power_controller shift enabled -min_switching_threshold_percentage 20
...
set_system_mode analysis
write_edt_files low_power_enabled
```

Given the compression logic created with the preceding command, use the following commands during ATPG phase to turn the power controller on with a 22% switching threshold for scan chain loading and to generate test patterns.

```
set_power_control shift -switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold, and creates the dofile and test procedure file that disable the hardware during pattern generation.

```
set_edt_power_controller shift disabled -min_switching_threshold_percentage 20
...
set_system_mode analysis
write_edt_files low_power_disabled
```

Given the compression logic created with the preceding command, use the following commands to enable the power controller and set it to a 22% switching threshold. This application also requires that you modify the edt_low_power_shift_en pin value in the test procedure file.

```
set_edt_power_controller shift enabled
set_power_control shift -switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```

Related Topics

[report_edt_configurations](#)
[set_power_control](#)
[report_power_metrics](#)

set_external_capture_options

Context: dft -edt, patterns -scan, patterns -scan_retargeting,
patterns -scan_diagnosis

Mode: setup (patterns -scan context only), analysis

Specifies the number of tester cycles to be used in the capture window of all patterns.

Usage

```
set_external_capture_options {OFF | -MINimum_cycles cycles | -FIXed_cycles cycles
| {-PLL_cycles cycles timeplate_name} } | {-CAPture_procedure procedure_name}
```

Description

Specifies the number of tester cycles to be used in the capture window of all patterns.

The set_external_capture_options command allows you to specify a minimum or fixed number of external tester capture cycles when saving -mode_external patterns for the current test session, including chain test patterns with capture cycle and retention test patterns. The integer you supply with this command specifies the minimum or fixed number of tester cycles that must be present between the end of the load_unload procedure and the start of the next pattern, or next load_unload procedure for multi-load patterns. When saving -mode_internal patterns for purposes other than pattern retargeting, the tool ignores set_external_capture_options settings and saves the patterns created by the ATPG engine, using the clock_sequential and capture procedures as is normally done.

For techniques to handle slow scan enable transitions, see “[Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions](#)” in the *Tessent Scan and ATPG User's Manual*.

Note

 The options you specify with the set_external_capture_options command apply to all pattern retargeting patterns, including mode_internal.

The integer you supply, for fixed or minimum, adds extra padding cycles to the end of the capture window if the number of tester cycles used in the pattern is less than the specified number when saving mode-external simulation or tester interface format patterns. The ASCII and binary format patterns are not affected by these padding cycles. The number of tester cycles you designate also applies to scan loads used in multi-load patterns, so the launch cycles used between scan loads in the multi-load pattern is also padded to ensure the same number of tester cycles between scan loads.

The padding cycle is created from a copy of the last cycle from the capture procedure with all clocks turned off, all measure data removed, and all pulse-always clocks turned on. This padding cycle is added after the last cycle of the capture procedure and before the first cycle of the next pattern.

When using **-PLL_cycles**, instead of an annotation that lists the number of padding cycles being added, there is an annotation that lists the original ATPG sequential depth for this pattern.

When STIL and WGL patterns are read back into the ATPG tools, the tool flags padding cycles added to the end of the capture window as added dummy cycles and does not use those padding cycles when converting the test cycles into ATPG pattern data. However, the tool uses these cycles when converting cycle-based fail information into pattern-based fail data.

As an alternative to directly specifying the number of external tester capture cycles, you can use the **-CAPture_procedure** switch and string pair to specify an external capture procedure that defines the external capture cycles to use with clock control definitions.

Issuing the command with no options reports the current settings. Any subsequent use of this command overrides all previous settings.

IDDQ Fault Type Usage Issues

If the fault type is set to IDDQ prior to using the **set_external_capture_options** command, and if you use the command to specify the **fixed_cycles**, **pll_cycles**, or **external_capture** option, the tool issues an error and does not set the option. This error is as follows:

```
// Error: The <option_name> option is not valid when generating IDDQ  
// patterns.
```

Arguments

- **OFF**
An optional literal that turns off any previous tester cycle settings. Note, however, that this option does not automatically disable maximum sequential depth. You have to issue **set_pattern_type –Max_sequential Off** to disable the setting for the maximum sequential depth.
- **-MINimum_cycles *cycles***
An optional switch and integer pair that specifies the minimum number of cycles present in the capture window when saving patterns in STIL or WGL format.
- **-FIXed_cycles *cycles***
An optional switch and integer pair that specifies the fixed number of tester cycles that needs to be present in the capture window when saving patterns in STIL or WGL format. The tool automatically calls “**set_pattern_type –Max_sequential**” to set the maximum sequential depth to be *integer* if the setting for maximum sequential depth is disabled or the maximum sequential depth is less than *cycles*.
- **-PLL_cycles *cycles timeplate_name***
An optional switch that specifies the fixed number of PLL capture cycles (integer value) and string that specifies the name of a timeplate that was loaded with the procedure file when saving patterns in STIL or WGL format. **PLL_cycles** can be used with **clock_control** definitions or with named capture procedures when they only have an internal mode. The

restrictions for PLL_cycles for the hold_pi and mask_po also apply to -CAPture_procedure as well.

This switch is intended to be used with clock_control definitions only. To use the switch, all clocks must either be controlled by clock_control definitions, be constrained, or be pulse-always or pulse-in-capture. In addition, hold_pi must be on, and mask_po must be on for all outputs. The -PLL_cycles switch replaces the ATPG cycles with a set number of external capture cycles using the specified timeplate. It is somewhat similar to internal_mode and external_mode of the named capture procedures. The internal_mode is what is generated by the ATPG engine using the above restrictions, and the external_mode is the number of cycles specified with the designated timeplate, and the clocks that pulse are the pulse_always or pulse_in_capture clocks. It is possible to specify the number of pll_cycles to be less than the max_sequential_depth setting. As with the -MINimum_cycles or -FIXed_cycles, -PLL_cycles is only used when saving -Mode_external patterns.

- -CAPture_procedure *procedure_name*

An optional switch and string pair that specifies the name of an external_capture procedure, which is used for all capture cycles between each scan load, even when the pattern is a multi-load pattern. For more information about the external_capture procedure, refer to the section “[Capture Procedures \(Optional\)](#)” in the *Tessent Shell User’s Manual*.

Examples

The following example adds one PLL capture cycle to the end of the capture window:

```
set_external_capture_options -pll_cycles 1 pll_timeplate
```

Related Topics

[set_pattern_type](#)

set_external_simulator

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.

Usage

```
set_external_simulator {-SIMulation_script script_filename} | {vsim [options...]}
```

Description

Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.

The command string must be valid for the simulator; therefore, you may need to refer to the simulator's user documentation for correct syntax and arguments.

Arguments

- **-SIMulation_script *script_filename***

A required switch and string pair that specifies a simulation script to use for verifying patterns. This script will be used automatically for the [analyze_simulation_mismatches](#) command. Supported simulation tools include vsim, ncverilog, and vcs.

For a script example, refer to this “[Example](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **vsim [*options...*]**

A required literal and optional string that specifies a basic invocation command for ModelSim in batch mode.

Examples

Example 1

The following example checks the current external simulator setting, then specifies to invoke ModelSim in batch mode with loading messages disabled, when verifying simulation results:

```
report_external_simulator
./vsim -c

set_external_simulator vsim -c -quiet
report_external_simulator
./vsim -c -quiet
```

ModelSim supports simulation with Standard Delay Format (SDF) back-annotation. Generally, you can apply the timing information in an SDF file to any instance in a design.

Example 2

The following example includes arguments that direct the simulator to annotate maximum timing values from the SDF timing file *cpu1.sdf* to instance /my_design/inst1, and minimum timing values from the SDF timing file *cpu2.sdf* to instance /my_design/inst2:

```
set_external_simulator vsim -c -sdfmax /my_design/inst1=cpu1.sdf
-sdfmin /my_design/inst2=cpu2.sdf
```

Example 3

The following example includes an argument that provides visibility into nets, ports, and registers while decreasing run time:

```
set_external_simulator vsim -c -voptargs+=acc=npr
```

The visibility provided by the “+acc=npr” option should be sufficient for debugging advanced test benches. For even more visibility but with less performance gain, you can modify the above example as follows:

```
set_external_simulator vsim -c -voptargs+=acc
```

Related Topics

[analyze_simulation_mismatches](#)
[report_external_simulator](#)
[report_measure_cycles](#)

[setfails_report](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the design rules checker displays clock rule failures.

Usage

`setfails_report Off | ON`

Description

Specifies whether the design rules checker displays clock rule failures.

The `setfails_report` command displays all clock rule failures of the design rules checker. The default mode upon invocation of the tool is Off.

Arguments

- **Off**

A literal that specifies for the design rules checker to not display clock failures. This is the default upon invocation of the tool.

- **ON**

A literal that specifies for the design rules checker to display clock failures.

Examples

The following example displays clock failures from the design rules checking process:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
setfails_report on
set_system_mode analysis
```

Related Topics

[add_clocks](#)

[delete_clocks](#)

[report_clocks](#)

set_failure_mapping_options

Context: patterns -failure_mapping

Mode: analysis

Specifies filters you can apply when performing reverse mapping of top-level failures to the core in hierarchical designs flows.

Usage

```
set_failure_mapping_options [-cycle_offset offset] [-short_core_name on | off]
```

Arguments

- **-cycle_offset *offset***

An optional switch and integer pair that specifies to adjust the cycles in the top-level cycle-based failure file by the value *offset*. In hierarchical flows, Tesson Diagnosis uses the adjusted cycles during reverse mapping to the core-level failure cycles/patterns. For more information, refer to “[Cycle Offset Adjustment for Failure Files](#)” in the *Tesson Diagnosis User’s Manual*.

- **-short_core_name on | off**

An optional switch and literal pair that specifies whether to truncate the names of core-level failure files so that they do not include the core instance pathnames of the files. At times, the addition of the core instance pathnames (the default behavior) can cause the filenames to exceed the maximum of 255 characters. When you turn this option on, the tool replaces the core instance pathnames with an index counter, “_*N*”, where *N* is an integer.

Description

For hierarchical design flows, use this command to adjust the way that Tesson Diagnosis performs reverse mapping of the top-level failures to the core.

[set_fault_mode](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the fault mode is collapsed or uncollapsed.

Usage

`set_fault_mode Uncollapsed | Collapsed`

Description

Specifies whether the fault mode is collapsed or uncollapsed.

The `set_fault_mode` command specifies whether the tool uses collapsed or uncollapsed fault lists for fault counts, test coverages, and fault reports. The default fault mode upon invocation of the tool is Uncollapsed. When you display a report on uncollapsed faults, the tool lists the representative fault first followed by its equivalent faults.

The DFTVisualizer Browser and Library Browser windows display fault information based on this setting.

Note

 This command is not valid for the bridge logic fault model (specified with the [set_fault_type](#) Bridge -Static command).

Arguments

- **Uncollapsed**

A literal specifying that the tool include equivalent faults in the fault lists. This is the default mode upon invocation of the tool.

- **Collapsed**

A literal specifying that the tool not include equivalent faults in the fault lists.

Examples

Example 1

The following example sets the fault mode to collapsed and then displays only the collapsed faults:

```
set_system_mode analysis
add_faults -all
set_fault_mode collapsed
report_faults -all
```

Example 2

The following shows an example when reporting uncollapsed tied faults as compared to reporting collapsed tied faults:

Uncollapsed:	Collapsed:
0 TI /I_140/I	0 TI /I_140/I
1 TI /II_140/O	1 TI /II_140/O
1 EQ /II_140/I	

Related Topics

[add_faults](#)
[delete_faults](#)
[read_faults](#)
[report_faults](#)
[report_testbench_simulation_options](#)
[set_fault_sampling](#)
[set_fault_type](#)
[write_faults](#)

[set_fault_sampling](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the fault sampling percentage used for circuit evaluation or scan identification.

Usage

```
set_fault_sampling {Off | percentage} [-Seed integer]
```

Description

Specifies the fault sampling percentage used for circuit evaluation or scan identification.

Fault sampling enables you to process a fraction of the total faults to decrease processing time for evaluating large circuits. For faults other than bridge faults, once you specify a percentage, the tool randomly picks the fault samples to process. For bridge faults, the tool randomly picks bridges to process; if a particular bridge is included in a bridge sample, all faults associated with that bridge are included.

Arguments

- **Off**

A switch that turns off fault sampling. This is the invocation default.

- *percentage*

A required, positive number that specifies the fault sampling percentage used for circuit evaluation or scan identification. Any number greater than 0 and less than or equal to 100 can be specified. The default is 100.

- -Seed *integer*

A switch and hex number pair that specifies a seed value to use in the selection of fault samples. Specifying unique seed values for different runs can provide more accurate fault sampling results. The number value must be a lower case, 32-bit hex representation. 0 is not a valid seed value. The default is 0XCCCCCCCC.

Examples

Example 1

The following example performs a trial ATPG run in Tessent FastScan using a fault sampling of one percent of the total faults. Because the set_fault_sampling command overwrites parts of the

original fault list, after the test run the example restores the fault list to the state it was in prior to the set_fault_sampling command:

```
set_system_mode analysis
add_faults -all
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
add_faults -all
```

Example 2

The following example performs the trial ATPG run, as in the preceding example, but uses faults from an external fault file named my_fault_file. The example then recovers from the set_fault_sampling command by clearing the internal fault list and reloading faults from the external file:

```
set_system_mode analysis
read_faults my_fault_file
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
read_faults my_fault_file
```

Related Topics

[add_faults](#)
[read_faults](#)
[report_faults](#)
[set_fault_mode](#)
[set_fault_type](#)
[write_faults](#)

set_fault_subclass_analysis

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Turns on or off the analysis for the specified fault class(es).

Usage

```
set_fault_subclass_analysis {-Enable | -Disable} {fault_class ... }
```

Description

Turns on or off the analysis for the specified fault class(es).

The **set_fault_subclass_analysis** command supports all of the fault classes and sub-classes supported by the [set_relevant_coverage](#) command, as well as the DI.EDT fault sub-class. To enable the analysis of a class of faults, you must issue this command *before* adding the faults.

By default, AU and DI sub-classes analysis are enabled; in fact, there is no way to disable any DI sub-class other than DI.EDT. By default, the DI.EDT fault sub-class is enabled unless you have turned off EDT Finder (in which case, you cannot enable the DI.EDT fault sub-class). You can override this default and disable DI.EDT analysis. For more information about EDT Finder, refer to the [set_edt_finder](#) command.

You can use the [set_relevant_coverage](#) command to add or remove the AU and UU fault classes from the relevant test coverage reported by the tool. The UU sub-class is automatically enabled using the [set_relevant_coverage](#) command as shown in “[Example 2](#)” on page 2123. When enabled, the UU sub-class analysis is performed only for stuck, transition, and bridge fault types.

Note that AU.EDT and AU.FP sub-classes are enabled by default only if the tool can identify the EDT-related faults and false path faults. The tool can identify EDT-related faults if the EDT logic was generated with the default name or if you specified the naming of the EDT module. The tool can identify false path faults if you defined them using the [add_false_paths](#) command, or if you read the SDC file into the tool.

For more information about fault sub-classes, refer to “[Fault Sub-classes](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **-Enable**
Switch that enables analysis for the specified fault class.
- **-Disable**
Switch that disables analysis for the specified fault class. Note that there is no way to disable any DI sub-class other than DI.EDT.

- *{fault_class ... }*

Repeatable string that specifies the fault class or the DI.EDT fault sub-class. [Table 6-18](#) on page 2346 lists the fault classes that can be specified with this command. You can specify a fault class either by name or code.

Examples

Example 1

The following example shows the [report_statistics](#) output when UU faults are classified into two sub-classes. Note, the `set_fault_subclass_analysis` command is issued first to enable UU analysis prior to adding faults.

```
SETUP> set_fault_subclass_analysis -enable UU
SETUP> set_system_mode analysis
...
ANALYSIS> add_faults -all
ANALYSIS> report_statistics

                                Statistics Report
                                Stuck-at Faults
-----
Fault Classes                               #faults
                                            (total)
-----
FU (full)                                 144
-----
UC (uncontrolled)                         4 ( 2.78%)
UU (unused)                               14 ( 9.72%)
AU (atpg_untestable)                     126 (87.50%)
-----
Fault Sub-classes
-----
UU (unused)
    CON (connected)                      1 ( 0.00%)
    FL   (floating)                      14 ( 9.72%)
AU (atpg_untestable)
    TC* (tied_cells)                   126 (87.50%)
*Use "report_statistics -detailed_analysis" for details.
-----
Coverage
-----
test_coverage                            0.00%
fault_coverage                           0.00%
atpg_effectiveness                      97.22%
-----
#test_patterns                           0
#simulated_patterns                     0
CPU_time (secs)                          2.7
-----
```

Example 2

The following example shows how to include UU.CON faults in relevant test coverage, which automatically enables UU faults for sub-class analysis:

```
set_relevant_coverage -include UU.CON
set_relevant_coverage
```

```
Relevant Coverage Setting Report
-----
Fault Subclass          Relevant Coverage
-----
AU.BB                  yes
AU.EDT                 no
AU.PC                  yes
AU.TC                  yes
AU.CC                  yes
AU.FP                  no
AU.MCP                 yes
UU.FL                no
UU.CON               yes
TI                     no
BL                     no
RE                     no
-----
```

Related Topics

[report_statistics](#)
[set_relevant_coverage](#)

set_fault_type

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the fault model type for ATPG.

Usage

```
set_fault_type {  
    Stuck  
    | TOGGLE  
    | {Iddq [-NO_SEquential_inputs_blocked | -SEquential_inputs_blocked]  
        | [-ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFF}]}  
    | {Transition [-ALLOW_Shift_launch]  
        [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFF}]}  
    | {Path_delay [-Mask_nonobservation_points] [-ROBust_detection_only]  
        | [-NO_Functional_detection] [-HAZARD_free_robust_detections]}  
    | Bridge  
    | {Udfm [-STATIC_faults]}  
    | {Udfm -DELAY_faults [-ALLOW_Shift_launch]  
        [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFF}]}  
    | {Udfm -IDDQ_faults [-NO_SEquential_inputs_blocked | -SEquential_inputs_blocked]  
        | [-ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFF}]}  
}
```

Description

Specifies the fault model type for ATPG.

The fault sites of all models are the input and output pins of the design cells in addition to external pins. 0 and 1 values are used to indicate the type of fault at the fault site for all fault models. Each fault model has its own separate fault collapsing according to the model's rules of equivalence.

When you change the fault type, both the current fault list and the internal pattern set are deleted.

Use the [set_multiple_detection](#) command to set the number of detections for each fault.

For more information on the different fault models, refer to the [Tessent Scan and ATPG User's Manual](#).

Arguments

- **Stuck**

A literal that specifies the single stuck-at fault model. This is the default upon invocation of the tool.

- TOGGLE

A literal that specifies the toggle fault model. The tool supports multiple detection for the toggle fault type. If you are using a legacy simulator, the tool will issue an error to indicate that it does not support toggle multiple detection.

- Iddq [-NO_SEquential_inputs_blocked] -SEquential_inputs_blocked | -ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFF}]

Optional literal and switch pair that specifies the Iddq fault model. Optional switches that control the treatment of Iddq faults include:

-NO_SEquential_inputs_blocked — An optional switch that specifies a pseudo-stuck-at model and prevents *a priori* categorization of faults as BLocked, which must propagate through sequential primitives. This can allow creation of test patterns that would not otherwise be created, and can improve defect coverage. The additional tests created can improve defect coverage, especially for Iddq-detectable defects in scan cells. This is the default.

For the pseudo-stuck-at model, many faults are marked BLocked during pre-analysis. This is done for cells where the tool cannot predict the transistors inside the cell and cannot know how to observe to a cell output, such as a Ram cell. Such faults are not considered part of the fault population and do not show as undetected for test_coverage.

-SEquential_inputs_blocked — An optional switch that specifies a pseudo-stuck-at model and that retains historical behavior of *a priori* categorization of faults as BLocked which must be observed by propagation into or through a sequential primitive, such as a SL in a scan cell.

For the pseudo-stuck-at model, many faults are marked BLocked during pre-analysis. This is done for cells where the tool cannot predict the transistors inside the cell and cannot know how to observe to a cell output, such as a Ram cell. Such faults are not considered part of the fault population and do not show as undetected for test_coverage.

-ALLOW_IDDQ_MEASURE_when_clock_active ON | OFF — An optional switch and literal pair that specifies the tool is allowed to measure Iddq when the clock(s) is active while generating Iddq patterns. This switch is on by default.

Note

 The -no_sequential_inputs_blocked, -sequential_inputs_blocked, and -allow_iddq_measure_when_clock_active switches behave the same for the UDFM fault model as for IDDQ faults.

The tool supports multiple detection for the IDDQ fault type. If you are using a legacy simulator, the tool will issue an error to indicate that it does not support IDDQ multiple detection.

- **T**ransition

A literal that specifies the transition fault model. The default behavior prevents the tool from creating transition tests that launch off of the last shift when it is unable to generate clock sequential tests. It will only consider generating transition patterns that launch in some cycle after the shift is completed (broadside).

Note

 By default, the transition option does not prevent the tool from asserting the scan enable signal during at-speed tests. You must use other means (scan_enable pin constraint or capture procedure, for example) to prevent inappropriate assertion of the scan enable pin during the at-speed part of a multiple cycle sequential transition fault test.

The default behavior described applies to “udfm -delay_faults” and transition fault types.

- **-ALLOW_Shift_launch**

An optional switch, available for transition faults and “udfm -delay_faults”, used to enable the generation of native launch-off-shift (LOS) patterns where the transition is launched by the last shift cycle of the scan load operation. It is only practical to use when the clock(s) are controlled directly from the primary inputs and not from an on-chip source such as a PLL. Since using a PLL as the internal clock source for at-speed test is very common, you will rarely use this switch.

Even without using the -allow_shift_launch switch, a test pattern may launch a transition fault through a shift operation. For example, the tool may generate a 2-cycle transition test pattern with the scan_enable asserted in the first capture cycle to launch the transition of the targeted fault and de-asserted in the second cycle for capture. This is referred to as pseudo-LOS. Unless you have designed your scan_enable to allow at-speed transition of this signal (for example, through pipelining), you will normally not want ATPG to generate this type of pattern. To prevent generating pseudo-LOS patterns, you need to constrain scan_enable to 0.

- **-APPLY_HOLD_PI_attribute_to_shift_launch ON | OFF**

An optional switch and literal pair that determines if the hold attribute defined at primary inputs for launch-off-shift patterns is used. The default for this switch is ON.

During test generation for launch-off-shift patterns, the tool forces primary inputs with hold attribute to have the same value as the value in the last shift cycle except for the following conditions:

- The PI is a clock pin or a read/write control pin.
- The PI is explicitly forced in the last shift cycle by load/unload and/or shift procedures and the forced value is not the same as the pin constraint value if there is any.
- All scan input pins will be treated as TIE-X during test pattern generation.

For these exception pins, the tool ignores the hold attribute defined at these pins.

- **Path_delay**

A literal that specifies the path delay fault model.

- **-Mask_nonobservation_points**

An optional switch that masks all non-observation points so path delay patterns are generated with values only at observation points. If there is already an internal pattern set in the tool and the fault type is already set to path delay, re-issuing the command with this switch alters the internal pattern response values by masking the non-observation points. If this switch was specified previously when setting the fault type to Path_delay, re-issuing the command without the switch causes the tool to re-simulate the internal patterns to restore the capture values at the non-observation points.

- **-ROBust_detection_only**

An optional switch that generates only robust tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test, but if unable to generate a robust test, will use a non-robust test or a functional test. This switch overrides the default behavior, forcing the tool to use only robust tests. If you include this switch, the tool will classify as ATPG_untestable (AU) all path delay faults that can only be tested non-robustly or functionally. For more information about robust, non-robust, functional, and hazard-free detections, refer to “[Path Delay Fault Detection](#)” in the *Tessent Scan and ATPG User’s Manual*.

Note

 This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- **-NO_Functional_detection**

An optional switch that disables the generation of functional tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test, but if unable to generate a robust test, will use a non-robust test or a functional test. This switch overrides the default behavior, forcing the tool to use robust tests or non-robust tests, but not functional tests. If you include this switch, the tool will classify as ATPG_untestable (AU) all path delay faults that can only be tested with functional tests. For more information about robust, non-robust, functional, and hazard-free detections, refer to “[Path Delay Fault Detection](#)” in the *Tessent Scan and ATPG User’s Manual*.

Note

 This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- **-HAZARD_free_robust_detections**

An optional switch that specifies that the tool holds off-path inputs of the gates in a stable state when it propagates the transition through the path. When creating robust path delay patterns, this switch provides a delay test with additional restrictions than when just using

the `-robust_detection_only` switch. This switch can be used with the `-robust_detection_only` switch to guarantee that any robust detection patterns created by the tool are hazard-free.

Hazard-free robust detections reduce the chance of glitches because the output arrival time is not hampered by the arrival time of the off-path input. For more information about robust, non-robust, functional, and hazard-free detections, refer to “[Path Delay Fault Detection](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **Bridge**
A literal that specifies the static bridge fault model.
- **Udfm**
Optional literal that specifies a user-defined fault model (UDFM).
- **-STATIC_faults**
Optional literal that enables static faults only. This is the default.
- **-DELAY_faults**
Optional literal that enables delay faults only. Optional switch details are described in the [Transition](#) fault model argument description.
- **-IDQ_faults**
Optional literal that enables IDQ faults only. Optional switch details are described in the [Idq](#) fault model argument description.

Examples

Example 1

The following example performs ATPG using the stuck fault model:

```
set_system_mode analysis
set_fault_type stuck
create_patterns
```

Example 2

The following example performs ATPG with the transition fault model and, by default, prevents the tool from generating basic combinational patterns that launch off the last shift:

```
set_system_mode analysis
set_fault_type transition
create_patterns
```

Example 3

The next patterns -scan example performs ATPG with the transition fault model, specifying to target three detections per fault during pattern generation, and to detect each DS fault three times during simulation before dropping it from the fault list. Data resulting from the multiple detection setting is highlighted in bold.

Tip

- i** When multiple detection is enabled, the number shown in the “#faults in list” column in the transcript is the cumulative tally of faults not yet detected the specified number of times. When multiple detection is not enabled, this column lists the number of faults not yet detected once. In either case, the “test cvrg” and “#faults detected” columns refer to faults detected at least once.
-

```
set_system_mode analysis
set_fault_type transition
set_multiple_detection -guaranteed_atpg_detections 3
set_pattern_type -sequential 2
create_patterns
```

```

//-----+
//          Analyzing the design |
//
//      Current clock restriction setting:      Clock_po
//      Calling: set_clock_restriction domain_clock -any_interaction
//
//      Current split capture setting:        Off (optimal)
//
//      Current clock off simulation setting: Off (optimal)
//
//      Current abort limit setting:         30
//      Calling:      set_abort_limit 300 100
//-----+
//
//      Current sequential depth:          0
//      Optimal sequential depth:        2
// Warning: Broadside transition testing requires minimum depth of 2.
// Calling: set_pattern_type -sequential 2
//
//-----+
// ATPG targeting 1 detection
//-----+
// Simulation performed for #gates = 250 #faults = 1080
// system mode = ATPG pattern source = internal patterns
//
//-----+
// #patterns test      #faults #faults # eff. # test BCE
// simulated coverage in list detected patterns patterns %
// deterministic ATPG invoked with comb/seq abort limit = 300/100
// --- ----- --- --- --- --- -----
// 64    92.03%    772    993    61    61    84.06%
// --- ----- --- --- --- --- -----
// 128   96.47%    668    46     16    77    90.05%
//-----+
// ATPG targeting 2 detection, starting with 77 patterns
//-----+
// Simulation performed for #gates = 250 #faults = 67
// system mode = ATPG pattern source = internal patterns
//
//-----+
// #patterns 2-det. #faults #faults # eff. # test BCE
// simulated test cov. in list detected patterns patterns %
// deterministic ATPG invoked with comb/seq abort limit = 300/100
// --- ----- --- --- --- --- -----
// 192   96.47%    510    67     28    105   93.40%
//-----+
// ATPG targeting 3 detection, starting with 105 patterns
//-----+
// Simulation performed for #gates = 250 #faults = 69
// system mode = ATPG pattern source = internal patterns
//
//-----+
// #patterns 3-det. #faults #faults # eff. # test BCE
// simulated test cov. in list detected patterns patterns %
// deterministic ATPG invoked with comb/seq abort limit = 300/100
// --- ----- --- --- --- --- -----
// 256   96.47%    409    69     25    130   94.71%
                                         Statistics Report
                                         Transition Faults
-----+
Fault Classes                      #faults

```

```

        (total)

-----
 FU (full)           1114
 -----
 DS (det_simulation) 1039   (93.27%)
 UU (unused)          30    ( 2.69%)
 TI (tied)             4    ( 0.36%)
 RE (redundant)         3    ( 0.27%)
 AU (atpg_untestable) 38    ( 3.41%)
 ----

Coverage
-----
 test_coverage          96.47%
 fault_coverage          93.27%
 atpg_effectiveness      100.00%
 -----
 #test_patterns            130
 #clock_sequential_patterns 130
 #simulated_patterns       256
 CPU_time (secs)           0.5
 ----

Multiple Detection Statistics
-----
 

| Detections<br>(N) | DS Faults<br>(Detection == N) | Test Coverage<br>(Detection >= N) |
|-------------------|-------------------------------|-----------------------------------|
| 1                 | 0 ( 0.00%)                    | 1039 ( 96.47%)                    |
| 2                 | 0 ( 0.00%)                    | 1039 ( 96.47%)                    |
| 3                 | 93 ( 8.35%)                   | 1039 ( 96.47%)                    |
| 4                 | 56 ( 5.03%)                   | 946 ( 87.84%)                     |
| 5                 | 54 ( 4.85%)                   | 890 ( 82.64%)                     |
| 6                 | 56 ( 5.03%)                   | 836 ( 77.62%)                     |
| 7                 | 50 ( 4.49%)                   | 780 ( 72.42%)                     |
| 8                 | 46 ( 4.13%)                   | 730 ( 67.78%)                     |
| 9                 | 54 ( 4.85%)                   | 684 ( 63.51%)                     |
| 10+               | 630 ( 56.55%)                 | 630 ( 58.50%)                     |


-----
 bridge_coverage_estimate          94.71%
 -----

```

Example 4

The next example performs ATPG with the path delay fault model and only robust detection patterns. The example then repeats the pattern creation run after enabling the tool to use both robust and non-robust detection patterns. Be aware that when you change the type of test the tool can use to detect path delay faults, the tool automatically resets all AU faults identified in the preceding pattern creation run.

```

set_system_mode analysis
set_fault_type path_delay -robust_detection_only
read_faults my_path_delay_faultlist
create_patterns
...
set_fault_type path_delay -no_functional_detection
create_patterns

```

Example 5

The following example generates launch-off-shift patterns by holding all PIs:

```
set_pattern_type -sequential 0 -ram_sequential off
add_input_constraints -hold -all
set_fault_type transition -apply_hold_pi_attribute_to_shift_launch on
create_patterns
```

Example 6

The following example sets the UDFM as the current fault type:

```
set_fault_type udfm
```

Related Topics

[add_faults](#)
[delete_faults](#)
[identify_redundant_faults](#)
[read_fault_sites](#)
[report_faults](#)
[set_fault_mode](#)
[set_multiple_detection](#)
[write_faults](#)

set_flattener_rule_handling

Context: unspecified, all contexts

Mode: setup

Specifies how the tool globally handles flattening violations.

Usage

```
set_flattener_rule_handling rule_id [Error | Warning | NOTE | Ignore] [Verbose | Noverbose]
```

Description

Specifies how the tool globally handles flattening violations.

The set_flattener_rule_handling command specifies the handling of the messages for net checking, pin checking, and gate checking. You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the initial invocation setting as specified in the following list of rules.

Each rules violation has an associated occurrence message and summary message. The tool displays the occurrence message only for either error conditions or if you specify the Verbose option for that rule. The tool displays the rule identification number in all rules violation messages.

Arguments

- ***rule_id***

A required, non-repeatable literal that specifies the identification of the exact flattening rule violations whose message handling you want to change.

The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

- Net flattening violations are described in sections “[FN1](#)” through “[FN9](#)”.
- Pin flattening violations are described in sections “[FP1](#)” through “[FP13](#)”.
- Gate flattening violations are described in sections “[FG1](#)” through “[FG8](#)”.

- **Error**

An optional literal that specifies for the tool to display both the error occurrence message and immediately terminate the rules checking.

If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Warning

An optional literal that specifies for the tool to display the warning summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rules violation.

If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOTE

An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Ignore

An optional literal that specifies for the tool to not display any message for the rule's violations. The tool must still enforce some rules and they must pass to allow certain functions to be performed later.

If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOVerbose

An optional literal that specifies for the tool to display the occurrence message only once for the rules violation and give a summary of the number of violations.

If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Verbose

An optional literal that specifies for the tool to display the occurrence message for each occurrence of the rules violation.

If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

Examples

The following example changes the handling of the FG7 flattening rule to warning and specifies that each occurrence should be listed:

```
set_flattener_rule_handling fg7 warning -verbose
```

Related Topics

[report_flattener_rules](#)

[set_drc_handling](#)

set_gate_level

Context: unspecified, all contexts

Mode: setup, analysis

Specifies the hierarchical level of gate reporting and displaying.

Usage

`set_gate_level Primitive | Design`

Description

Specifies the hierarchical level of gate reporting and displaying.

The `set_gate_level` command specifies the hierarchical gate level at which the tool operates. This includes the reporting and schematic display of gate information. Once you set the gate level, the tool processes all subsequent commands using the new gate level.

Whenever you issue a command which invalidates the flat model, the tool also invalidates the hierarchical gate display structure. This causes DFTVisualizer to clear the schematic view area. You can rebuild the hierarchical gate structure by creating a new flat model. To do so, either enter and exit the Setup mode, or use the [create_flat_model](#) command.

Arguments

- **Primitive**

A literal that specifies to display gate information at the built-in primitive gate level.

- **Design**

A literal that specifies to display gate information at the design library hierarchical gate level. These are the top-level cells of the design library which are instantiated in your design. This is the default upon invocation of the tool.

Note



DFTVisualizer shows gate identification numbers (ID#s) with displayed instances only if the design level is set to Primitive.

Examples

The following example sets the gate report level so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the Setup system mode):

```
set_system_mode analysis
set_gate_level primitive
set_gate_report error_pattern
report_gates i_1006/o
```

Related Topics

[create_flat_model](#)
[report_gates](#)
[set_gate_report](#)

set_gate_report

Context: all contexts

Mode: all modes

Specifies the information displayed by the report_gates command and in the DFTVisualizer windows.

Usage

When EDT is Off

```
set_gate_report Normal | SImulation_context | Trace | Error_pattern |
  {PATtern_index pattern_index [-Internal | -External] [-SCan_test | -CHain_test]} |
  VCD time_value | {Parallel_pattern pattern_number} | Fault_status | Test_data |
  TlE_value | SEq_depth_data | {Clock_cone pin_name} |
  {CAPTURE_PROCedure named_capture_procedure} |
  {Drc_pattern {{Test_setup | TEST_End} [{-Cycle | -Time} n1 [n2 | End]}} |
  {Load_unload | SHIft | SKew_load | SHADOW_Control | Master_observe |
  SHADOW_Observe |
  STate_stability | STABle_after_setup | STABle_Capture | STABle_Load_unload
  | STABle_Shift [time | -All]} } | {DELay_data {LOcal_delay_data
  | STATIC_Arrival_time | STATIC_Propagation_time | STATIC_Slack
  | STATIC_PAth_delay | ACTUAL_Arrival_time pattern_number
  [-Internal | -External] | ACTUAL_Propagation_time pattern_number
  [-Internal | -External] | ACTUAL_Slack pattern_number [-Internal | -External] |
  ACTUAL_PAth_delay pattern_number [-Internal | -External]} }
  [-Timing_exceptions {ON | OFF}] [-COnstrain_value {ON | OFF}] [-MAX_Fanouts
  {number | UNlimited}] [-CLOCK_domains_in_capture_cycles {ON|OFF}]
```

When EDT is On

```
set_gate_report Normal | SImulation_context | Trace | Error_pattern |
  {PATtern_index pattern_index [-Internal | -External] [-SCan_test | -CHain_test]} |
  {Parallel_pattern pattern_number} | Fault_status | Test_data | TlE_value |
  SEq_depth_data | {Clock_cone pin_name} |
  {CAPTURE_PROCedure named_capture_procedure} |
  {Drc_pattern {{Test_setup | TEST_End} [{-Cycle | -Time} n1 [n2 | End]}}}
  | {Load_unload | SHIft | SKew_load | SHADOW_Control |
  Master_observe | SHADOW_Observe | STState_stability | STABle_after_setup |
  STABle_Capture | STABle_Load_unload | STABle_Shift [time | -All]} |
  {K19 [-Initial | -FIRst_random | -SECond_random |-Post_capture |
  -Low_power lp_sim#] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]} | {K22 [-Mask
  {No_mask | mask#}] [-All | Load_unload | {Shift [-All | cycle [cycle]]}] | F_rule_name}
  | DELay_data {LOcal_delay_data | STATIC_Arrival_time |
  STATIC_Propagation_time | STATIC_Slack | STATIC_PAth_delay |
  ACTUAL_Arrival_time pattern_number [-Internal | -External]
  | ACTUAL_Propagation_time pattern_number [-Internal | -External] |
  ACTUAL_Slack pattern_number [-Internal | -External]} |
```

ACTUAL_PATH_delay pattern_number [-Internal | -External]}}
[-Timing_exceptions {ON | OFF}] [-COnstrain_value {ON | OFF}]
[-CLOCK_domains_in_capture_cycles {ON|OFF}] [-MAX_Fanouts {number | UNlimited}]

For dft -scan

```
set_gate_report Normal | Trace | Error_pattern | TlE_value | {Drc_pattern {{Test_setup  
[{{-Cycle | -Time} n1} [n2 | End]]}} | Load_unload | SHIft | SKew_load |  
SHADOW_Control | Master_observe | SHADOW_Observe | STate_stability  
| STABle_after_setup | STABLE_Capture | STABLE_Load_unload | STABLE_Shift}  
[-All | time]} [-CLOCK_domains_in_capture_cycles {ON|OFF}]  
[-MAX_Fanouts {number | UNlimited}]
```

Description

Specifies the information displayed by the report_gates command and the DFTVisualizer schematic windows.

When you exit the Setup system mode, the trace and any rules-checking error pattern results will not be available with this command. For information on the format output by the different options in this command, refer to the [report_gates](#) command description.

The data field of the report created by the report_gates command displays one character for each event within the cycle.

Note

 When EDT is On, the tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, report_gates cannot report simulation values for the EDT logic during these cycles regardless of the option set with the set_gate_report command.

Arguments

- **Normal**

A literal that specifies for the report_gates command to display only its standard information. This is the default mode upon invocation of the tool.

- **simulation_context**

A literal that specifies for the report_gates command to display the simulated values from the current simulation context. Refer to “[Example 11](#)” on page 2169.

- **Trace**

A literal that specifies for the report_gates command to display the simulated values of the gates during the scan chain tracing. The format of these values depends on the contents of the shift procedure. If the shift procedure contains additional frames, the additional frames will also be displayed in the gate report data. The trace data relates to the simulation performed during the scan chain tracing. Use the Trace option to determine why a scan chain was not properly sensitized during the shift procedure.

Using this option will display scan path “don’t cares” as an S in the gate report to distinguish them from non-scan path “don’t cares” which display as X.

For an example of the information displayed, refer to “[Example 2](#)” on page 2156.

- **Error_pattern**

A literal that specifies for the report_gates command to display the simulated value of the gates and its inputs for the pattern at which an audit error occurred.

- **PATtern_index pattern_index [-Internal | -External] [-SCan_test | -CHain_test]**

A literal, integer, and optional switch triplet that specifies the pattern the [report_gates](#) command uses when displaying the value of a gate. The *pattern_index* must be a non-negative integer.

When reporting a sequential element, the gate report also displays in a pair of brackets ([]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

Glitch Reporting

When a leading-edge DFF clock is gating a leading-edge DFF structure, the tool displays the glitch on the CLK line as 0(1)00 and lets the downstream DFF capture. For example:

```
//  /sff9_pulse_inter (366)  DFF
//    "S"      I  (000-000)  188-
//    "R"      I  (000-000)  32-
//    CLK     I  (010-0(1)00)  298-/AND02_pulse_inter/Y
//    "D0"    I  (000-111)  284-
//    "OUT"   O  (100-011-[1])  152-  153-
```

The glitch 0(1)00 on the clock only shows up as the input pin values of the state element. The glitch does not show up as the output value of the clock driver.

False Path Reporting

The tool indicates in the pattern report if a false path is defined. One possible example is “1(X0)0”. The (X0) values in 1(X0)0 indicate that at the beginning of the second frame (leading edge frame), the value is unstable. The instability is due to the hold time effect, and the X occurs due to the leading edge (LE) clock event on the false path source flop FF. Because the LE clock event occurs at the second frame, (X0) are to indicate the value these are the values at the LE frame.

Primitive or Design Level Gate Reporting

Depending on whether the tool is set up for primitive or design level gate reporting, you may see additional information as follows (use the [report_environment](#) command to check the current gate level):

- Primitive level ([set_gate_level](#) Primitive):

If the sequential element you report on is part of a scan cell and the value captured by the element is not what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value and explanatory text is included indicating which element *is* observed for that scan cell and that pattern. “Unobs” and extra explanatory text is displayed, for example, if you report on the slave latch within an LSSD scan cell and the test procedure file includes a master_observe procedure (the slave is not observed when a master_observe procedure exists).

Note

 When reporting gates at the primitive level, the tool displays captured/unload values only for the outputs of DFF and LA primitives, both scan and non-scan. Captured/unload values are not reported for transparent latches (TLAs).

- Design level ([set_gate_level](#) Design):

If the sequential element you report on is a scan cell and an output value resulting from capture does not correspond to what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value. Explanatory text is not displayed for these “Unobs” values.

For examples of captured/unload displays for different scan cells and reporting levels (primitive or design), see the Examples section.

If, after setting the gate report with this option, you alter the tool environment, be sure to reissue the “set_gate_report” command. This will cause the tool to re-simulate the patterns, ensuring the values reported for a gate are up to date with the latest tool settings.

You can select from four pattern sources as specified here:

- Internal — An optional switch that specifies use of the tool’s internal pattern set. This is the default.
- External — An optional switch that specifies use of the tool’s external pattern set.
- Scan_test — An optional switch that specifies the report of the scan test pattern.
- Chain_test — An optional switch that specifies the report of the chain test pattern.

Together with the *pattern_index* argument, your option selection specifies the exact pattern to be reported as shown in [Table 6-9](#).

Table 6-9. Pattern Sources

Display Pattern Type	scan_test	chain_test
internal	internal scan pattern (default)	internal chain pattern

Table 6-9. Pattern Sources (cont.)

Display Pattern Type	scan_test	chain_test
external	external scan pattern	external chain pattern

- **VCD *time_value***

A literal and numerical pair that sets the gate report source to the VCD file used with the [analyze_simulation_mismatches](#) command. The Verilog simulation values after the specified simulation *time_value* can be seen using the [report_gates](#) command.

The possible reported values are:

- 0 — Logic 0
- 1 — Logic 1
- Z — Tristate
- X — Unknown
- U — Unused
- . — No VCD data

- **Parallel_pattern *pattern_number***

Note

 Use this option only if another command directs you to use it; for example, the [analyze_bus](#) command might transcript a message that explicitly says to use “set_gate_report parallel_pattern 0”. If you use this option, be sure [report_gates](#) is the first command you enter after simulation; bogus simulation values may be reported if you enter any other tool command after simulation, but before you [report_gates](#). The *Pattern_index* argument is the recommended way to specify a pattern for the [report_gates](#) command to use when displaying the value of a gate.

A literal and integer pair that specifies the pattern number from the last simulation pass that you want the [report_gates](#) command to use when displaying the value of a gate. The *pattern_number* must be an integer between 0 and 63.

When reporting a sequential element, the gate report also displays in a pair of brackets ([]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

- **Fault_status**

A literal that specifies for fault detection status of all gates to be preserved. Subsequent commands that cause the gate to be displayed will annotate the pins with fault status data. If a schematic is currently displayed in a DFTVisualizer schematic window and you change the gate report data (by issuing the [set_gate_report](#) command), all fault sites will be annotated with their fault detection status.

Note that this switch is not valid for the bridge logic fault model.

The format of the fault status data is as follows:

<sa0-status:sa1-status> — for stuck-at faults

<slow_to_rise-status:slow_to_fall-status> — for transition faults

where sa0-status and sa1-status are one of the following:

DS — Detected by simulation

DI — Detected by implication

PU — Possible detect untestable

PT — Possible detect testable

AU — Atpg untestable

UC — Undetected uncontrolled

UO — Undetected unobserved

UU — Untestable unused

BL — Untestable blocked

TI — Untestable tied

RE — Untestable redundant

F — Recognized fault site, but no fault has been added yet

N — Site isnofaulted; either because the fault site is inside a library model and internal faults are off, or because a user hasnofaulted it with the [add_nofaults](#) command

“-” — Nothing known about this pin; used for pins created by the flattening process or pins that are not fault sites (for example, the pins of an unnamed internal instance of a library cell)

When you use the Fault_status option, subsequent [report_gates](#) commands display the fault sub-class for untestable faults (AU/UC/UO). For a complete list of these fault sub-classes, see [Table 6-18](#).

The DFTVisualizer command, [report_display_instances](#), will report the fault detection status in the tool transcript.

- **Test_data**

A literal that specifies for the [report_gates](#) command to display previously-calculated control and observe values. To set the Test_data option, you must do so prior to rules

checking, and you must re-execute the design rules checking process; otherwise, no data (-) from this option will be available for gate reporting.

This option is primarily for logic BIST purposes (when inserting control and observation points). The data for each pin of a reported gate consists of three integers indicating how many times the pin was controllable to 0, how many times it was controllable to 1, and how many times it was observable during the preceding analysis. The data is displayed in the following format:

(# of times controlled to 0-# of times controlled to 1, # of times observed)

Note

 When the Test_data option is in effect, the [report_environment](#) command will show “BIST data” as the current gate report setting.

- **TIE_value**

A literal that specifies for the report_gates command to display the simulated values that result from all natural, tied gates and learned constant value non-scan cells.

Tie_value is available at any time the flat model exists. The value displayed on the TIE node is set by simulation. All internal nodes are initially set to X for simulation. Subsequently, ATPG or various commands may put values on the nodes. Occasionally, a set of circumstances may occur where an X shows up on a TIE node.

During operation, ATPG typically marks for simulation only those gates which are of interest for sensitization and propagation of a particular fault. If a gate is not marked, it will not be simulated, and any nodes associated with it may remain at an “X” value, even though they are connected to a TIE gate.

While tie_value reports the status of node values once it comes out of the DRC checks, the constrain_value tells you what the gate is suppose to act like for future consideration.

- **SEQ_depth_data**

A literal that specifies for the report_gates command to display the calculated gate sequential depths.

When you select a non-zero sequential depth, the tool performs a learning process to identify the minimum depths necessary to satisfy controllability and observability requirements for all gates using the current clock_sequential cells. The tool calculates both the 0-state and 1-state controllability depths. At the end of the analysis, the tool displays a summary message that indicates the largest sequential test depth, along with the largest control and observe depth. The test generator uses the sequential depth information in making decisions and avoiding paths whose sequential depth exceeds the maximum allowed sequential depth.

The report_gates command includes three values separated by a comma and a dash. The first value is the 0-state controllability depth, the second value is the 1-state controllability depth, and the third value is the observability depth. The maximum reported depth is 255. If

controllability or observability is logically impossible or exceeds 255, the report displays an asterisk (*) in the corresponding fields.

- **Clock_cone pin_name**

A literal and string pair that specifies the clock pin for which the report_gates command displays the clock cone data.

The clock cone data from the report_gates command is the same data that is available as error data for clock rules violations. You can only use this option after flattening the simulation model using the [create_flat_model](#) command.

The *pin_name* must be a valid clock pin or an error condition occurs. For example:

```
set_gate_report clock_cone {{\atpgclk[0] }}
```

The tool considers the pin equivalents when calculating the clock cones. State elements that the tool identifies as capturing on the clock's trailing edge will not propagate the clock effect cone. During the Setup system mode, this information is not available and the tool assumes all state elements capture with the leading edge of the selected clock.

When you try to report gates with the *clock_cone* source selected and the clock cone data is no longer present, the tool issues a warning message and sets the gate report source to the default before executing the command. This can happen, for instance, when you re-read the design or issue another command that causes the flat model to be regenerated. This is the warning:

```
// Warning: The precomputed clock cone data is no longer available.  
// Resetting the gate reporting mode.  
// Please reissue the set_gate_report command to see clock  
// cone data.
```

- **CAPTURE_PROCedure named_capture_procedure**

A literal that reports the value at each gate implied by forced and conditional assignments defined in the named capture procedure (NCP). This is useful for debugging NCP-specific issues like the inability to detect a fault or prevent bus contention.

- **Drc_pattern procedure_name [-All | time]**

Two literals and an optional time triplet that specifies the name of the procedure and the time in the test procedure file that the report_gates command uses to display a gate's simulated value.

The valid literal choices for the *procedure_name* option (for use with Drc_pattern) are as follows:

Test_setup — A literal that specifies the use of the test_setup procedure. In the test procedure file, this procedure sets non-scan elements to the state you desire for the **load_unload** procedure. The tool uses the entire test_setup procedure unless you restrict the report to a certain portion using the -Cycle or -Time switch. In order to conserve screen width, time values are listed vertically in Test_setup gate reports.

TEST_End — A literal that specifies the use of the test_end procedure. In the test procedure file, test_end is used to add a sequence of events to the end of a test pattern

set. The tool uses the entire test_end procedure. In order to conserve screen width, time values are listed vertically in test_end gate reports.

-Cycle — A switch that indicates *n1* is a cycle number and specifies to start the report at cycle *n1*.

-Time — A switch that indicates *n1* is a time and specifies to start the report at time *n1*. The time units are based on the timescale defined in the test procedure file, which by default is 1 nanosecond.

n1 — An integer that specifies a cycle or time at which to start reporting. When used with the -Time switch, *n1* specifies a time. When used with the -Cycle switch, *n1* specifies a cycle. The tool numbers cycles beginning with 0; so, for example, to specify the second cycle, you would use “-cycle 1”.

n2 — An optional integer that specifies to report from cycle (or time) *n1* to cycle (or time) *n2* and stop reporting.

End — An optional literal that specifies to report from cycle (or time) *n1* to the end of the test_setup procedure.

Note

 When using the -Cycle or -Time switch, if you do not include either the *n2* or End argument, gate reports will show data for only the *n1* cycle (or time).

Load_unload — A literal that specifies the use of the load_unload procedure. The test procedure file must contain this procedure, which describes how to load and unload data in the scan chains.

Shift — A literal that specifies the use of the shift procedure. The test procedure file must contain this procedure, which describes how to shift data one position down the scan chain.

Using this option will display scan path “don’t cares” as S in the gate report to distinguish them from non-scan path “don’t cares” which display as X.

The data in the report created by report_gates for shift data normally has three values, but if the procedure file defines four events, report_gates reports four values. One example may be when the force_pi occurs at some other time than at the beginning of the cycle. See [Example 15](#).

Skew_load — A literal that specifies the use of the skew_load procedure. In the test procedure file, this procedure describes how to propagate the output value of the preceding scan cell into the master memory element of the current cell (without changing the slave), for all scan cells.

SHADOW_Control — A literal that specifies the use of the shadow_control procedure. In the test procedure file, this procedure describes how to load the contents of a scan cell into the associated shadow.

Master_observe — A literal that specifies the use of the master_observe procedure. In the test procedure file, this procedure describes how to place the contents of a master into the output of its scan cell.

SHADOW_Observe — A literal that specifies the use of the shadow observe procedure. In the test procedure file, this procedure describes how to place the contents of a shadow into the output of its scan cell.

STate_stability — A literal that specifies the display of the simulation values for the **load_unload** procedure and the capture clock cycle that the tool used to determine the constant value state elements at the initial load time. The report separates the **shift** procedure values, **load_unload** procedure values, and the capture clock cycle with parentheses. This format provides information that is helpful when you are trying to debug boundary scan.

Note

 Only the simulation values for the first **load_unload** procedure that follows the **test_setup** procedure are displayed. Use the **load_unload** literal to display the final stable values in the **load_unload** procedure.

The **state_stability** option is not a procedure.

For detailed information about state stability reporting, refer to the “[State Stability Issues](#)” appendix in the *Tessent Scan and ATPG User’s Manual*.

STABLE_Capture — A literal that specifies the display of the values that are constant throughout the capture procedures.

0, 1, or Z — Indicates a constant value exists throughout the capture cycles.

X — Indicates a constant value does not exist throughout the capture cycles.

STABLE_Load_unload — A literal that specifies the display of the values that are constant throughout the **load_unload** cycles.

0, 1, or Z — Indicates a constant value exists throughout the **load_unload** cycles.

X — Indicates a constant value does not exist throughout the **load_unload** cycles.

STABLE_Shift — A literal that specifies the display of values that are constant throughout the shift cycles.

0, 1, or Z — Indicates a constant value exists throughout the shift cycles.

X — Indicates a constant value does not exist throughout the shift cycles.

STABLE_after_setup — A literal that specifies the display of the values that are constant in the period following the end of the **test_setup** procedure and the beginning of the **test_end** procedure. For scan patterns, the value is constant throughout scan and capture cycles. For ijtag patterns, the value is constant throughout any ijtag operations.

If a **load_unload** procedure exists, the values established at the end of **test_setup** are verified for stability using the **add_input_constraints -c1** and **c0** values which are also forced to a constant value during the **load_unload** procedure.

If a **load_unload** procedure does not exist, as in the ijtag context, stability analysis is only performed with the **add_input_constraints -c1** and **-c0** values.

0, 1, or Z — Indicates a constant value exists at the end of test_setup and is maintained for all subsequent events up to the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

X — Indicates a constant value does not exist in the period following the end of test_setup and the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

-All — An optional switch that specifies use of all times in the test procedure file. This is the default.

time — An optional positive integer, greater than 0, that specifies a time in the test procedure file.

This subsection describes Tessent FastScan and Tessent TestKompress gate reporting when Drc_pattern specifies a procedure.

When you set the gate report to a DRC pattern and procedure, most of the data reported by the report_gates command is simulation data regarding the specified procedure. For example, if you specify the load_unload procedure, you will get simulation data regarding the load_unload procedure immediately following the test_setup procedure. Statements like **apply shift** are broken out by surrounding ()s.

The last group of data is more specialized. Its contents depend on the capture clock being set with the -atpg switch. The starting state for this simulation results from simulating the events of the test setup procedure, followed by the load_unload procedure and its apply procedures (shift and shadow_control).

- Case 1: No Capture Clock

There will be 1 or 2 values in the last pair of ()s. The first value is the simulation state that results from holding all PIs at their pin constrained value and setting all clocks to X at the end of load/unload.

If any state element has a different binary value than the one it had at the end of simulating test setup, its value will be changed to X, the effect propagated, and the final values saved in the second value between the ()s. See the following examples:

```
timeplate gen_tp1 =
    force_pi 0;
    measure_po 100;
    pulse myclk 200 100;
    period 400;
end;
procedure shift =
    scan_group grp1;
    timeplate gen_tp1;
    cycle =
        force_sci;
        measure_sco;
        pulse myclk;
    end;
end;
procedure load_unload =
    scan_group grp1;
    timeplate gen_tp1;
    cycle =
        force scan_en 1;
        force myclk 0;
    end;
    apply shift 2;
end;
report_gates clk
// /CLK primary_input
//      CLK      O (0)(0X0)(0)(X)
```

The first (0) is the simulation of the events in the load_unload procedure prior to the apply shift, (0X0) is the simulation of the shift procedure, (0) is the simulation of the events in the load_unload after the shift, and finally (X) is the simulation of the clocks at X.

- o Case 2: Capture Clock -Atpg

There will be 3 or 4 values in the last pair of ()s. The first three values result from simulating a pulse of the capture clock with all other clocks set to the off value.

If any state element has a different binary value than the one it had at the end of simulation test setup, its value is changed to X, the effect is propagated, and the final values are saved in the fourth value between the ()s.

K19 [-Initial | -FIRst_random | -SECond_random | -Post_capture | -Low_power
lp_sim#] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]

Note

 Use this argument only when necessary. For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, the Drc_pattern argument will be sufficient and less time-consuming than this one.

A literal and an optional, procedure pair that specifies to include a gate's simulation data resulting from the K19 design rules check (DRC) in the output displayed by the report_gates command. If you specify "K19" without any options, *all* K19 simulation values for every event specified in the test procedure file are displayed except test_setup events.

Note

 The test_setup simulation data is available after you use the "set_gate_report drc_pattern test_setup" command.

The options enable you to limit the content of the display to the data for a certain procedure and/or for a certain cycle or range of cycles. For a detailed explanation of the K19 DRC, refer to "[K19](#)" and to "[Understanding K19 Rule Violations](#)" in the Tesson TestKompress User's Manual.

K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K19 reporting when you need to see the specific simulation values applied in each shift cycle. This is not commonly required as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

You must set this option in Setup system mode. Also, DRC handling for the K19 rule must be set to either error (the default) or warning, but not to ignore.

The following three switches specify to report simulated and emulated (where applicable) values for selected gates as described here:

- Initial — An optional switch that specifies to report the very first K19 simulation result.
- FIRst_random — An optional switch that specifies to report the first random initialization K19 simulation result.
- SECond_random — An optional switch that specifies to report the second random initialization K19 simulation result.
- Post_capture — An optional switch that specifies to report the second K19 simulation result (after capture); if low-power hardware exists and is enabled, its first simulation result is reported.
- Low_power lp_sim# — An optional switch that specifies to report the first or second simulation result for the low-power hardware, if one exists and is enabled. The lp_sim# argument can have a value of either 1 or 2.

The following three switches specify the use of the events in the test procedure file.

-All — An optional switch that specifies use of all events in the test procedure file except test_setup events. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle — A positive integer corresponding to a particular shift cycle. If you specify a single cycle number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all cycles in the range. Cycles are counted from 1 and include independent shifts.

K22 [-Mask {No_mask | mask#}] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]

Note

 Use this argument only when necessary. For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, the Drc_pattern argument will be sufficient and less time-consuming than this one.

An optional literal and procedure pair that include the gate simulation data resulting from the K22 design rules check (DRC) in the output displayed by the report_gates command. If “K22” is specified without any options, the K22 simulation values for every event specified in the test procedure file display. The options enable you to limit the content of the display to just the data associated with a certain procedure and/or to a certain cycle or range of cycles. For a detailed explanation of the K22 DRC, refer to “[K22](#)” and to “[Understanding K22 Rule Violations](#)” in the Tessent TestKompress User’s Manual.

K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K22 reporting when debugging incorrect compactor mask settings for specific patterns simulated by the design rule check. This is not commonly required, as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

Note

 The K22 evaluations of compactor gates are synchronized with measure_sco statements in the test procedure file.

You must set this option in Setup system mode. Also, DRC handling for the K22 rule must be set to either error (the default) or warning.

-All — An optional switch that specifies use of all events in the test procedure file. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle — A positive integer corresponding to a particular shift cycle. If you specify a single cycle number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all the cycles in the range. Cycles are counted from 1 and include independent shifts.

-Mask — An optional switch that sets the masking mode for the simulation data.

No_mask — A literal that together with -Mask specifies to report only non-masking mode simulation data.

mask# — A positive integer that, together with -Mask, specifies the identification number of a mask. The tool assigns a number to a mask according to the core scan chain the mask enables the tool to observe. Mask 3, for example, enables observation of the third core scan chain. This option thus limits reporting to the simulation data for the scan chain made observable by the specified mask.

F_rule_name — A literal that specifies an F rule between F1-F21.

Note, you must specify a specific occurrence of a F20 violation using the string “F20-occurrence#” where occurrence# is the violation occurrence. For more information, see the [report_drc_rules](#) command.

- **DEDelay_data [delay_data_option]**

A required literal pair that specifies to report delay data when using the [report_gates](#) command. For complete information, refer to “[Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual*.

Reported delay data is one of the following data types:

Number — Indicates delay data values.

STAT (Static) — Indicates that no transition is possible on the pin so a delay value is not calculated or reported.

NA (Not Applicable) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Static timing information)

- (dash) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Actual timing information) For the *local_delay_data* literal, a dash (-) is reported in the case of a conditional IO path delay; in this case, the timing information is included in the report as shown in [Example 9](#).

The following *delay_data_option* literals are available:

LOCAL_delay_data

Reports the local delay data, including the wire delay (INTERCONNECT) and gate delay (IOPATH) associated with the flattened gate pin.

STATIC_Arrival_time

Reports the longest static rising and falling arrival time of the gate and direct fan-ins.
Both minimal and maximum arrival time will be reported as the following format:

(max_rising_arrival_delay, max_falling_arrival_delay) /

(min_rising_arrival_delay, min_falling_arrival_delay)

STATIC_Propagation_time

Reports the longest static rising and falling propagation time of the gate and direct fan-ins.

STATIC_Slack

Reports the static test slack of the gate and direct fan-ins. The test slack is computed as the difference between the test clock period and the summation of longest arrival time and longest propagation time. The format of the reported data is:

(test_slack_of_rising, test_slack_of_falling)

STATIC_Path_delay

Reports the longest static rising and falling path delay time through the gates.

ACTUAL_Arrival_time [-Internal | -External]

Reports the actual arrival time of the gate and direct fan-ins. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum arrival time of given cycle will be reported. In addition to the actual arrival time, the simulation value of the specified pattern is also reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

This is the default for Tesson TestKompress and Tesson FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Propagation_time [-Internal | -External]

Reports both the actual propagation time and simulation propagation time. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum propagation time of given cycle will be reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

This is the default for Tesson TestKompress and Tesson FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Slack [-Internal | -External]

Reports both the actual and simulation slack values. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum slack of given cycle will be reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Path_delay [-Internal | -External]

Reports both the actual and simulation path delay through the gates. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum path delay among all cycles will be reported.

- Internal — An optional switch that specifies use of the tool's internal pattern set.
- External — An optional switch that specifies use of the tool's external pattern set.

Note

 The calculations of arrival time and propagation time assume that all PIs change value at time 0. The delays at the clock port and the setup time of the sequential elements are ignored.

- -Timing_exceptions ON | OFF

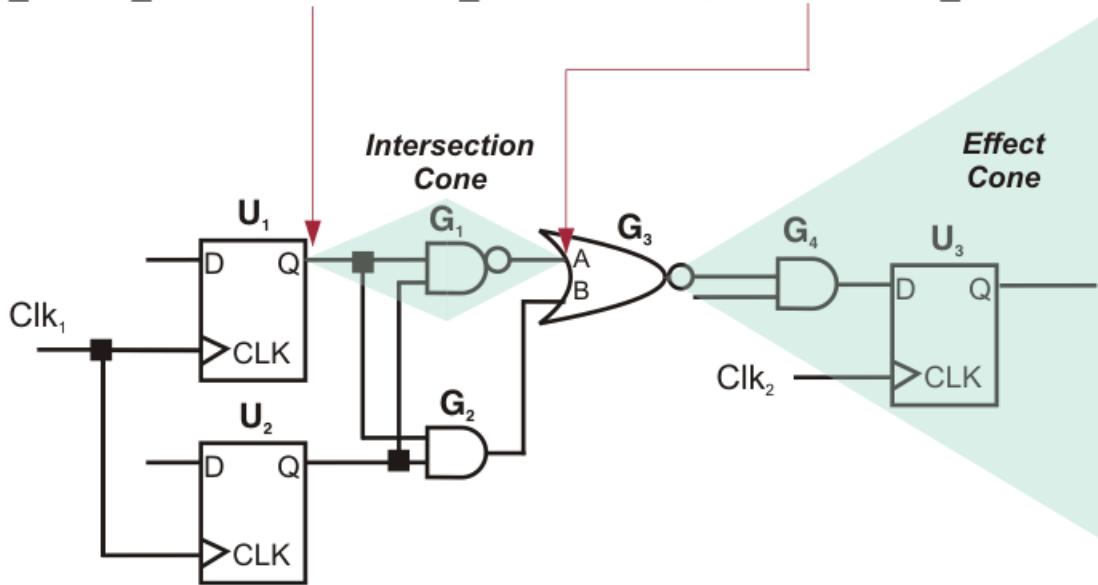
An optional switch and literal that turns on or off the timing exceptions (false paths or multicycle paths) information reporting for the [report_gates](#) command and the DFTVisualizer schematic windows. This option is only effective during non-Setup mode. The default is off.

[Figure 6-5](#) illustrates the false path definition point and the false path intersection cone and an effect cone.

Figure 6-5. False Path Intersection and Effect Cones

SDC File contains:

```
set_false_path -from [get_clocks {Clk1}] -to [get_pin {G3/A}]
```



In this figure, the false path begins from flops in clock domain Clk₁ to internal pin G₃/A. The false path's intersection cone (G₁) is the logic between all **-from** points and the **-to** point. The false path's effect cone (G₄ and U₃) is the combinational fanout cone of the **-to** point.

When you set the -Timing_exceptions switch to ON, any subsequent report_gates command can include one or more of the following false or multicycle path-related keywords in the report:

- **Fr** — Indicates the gate is a false or multicycle path from point.
- **To** — Indicates the gate is a false or multicycle path to point.
- **Th** — Indicates the gate is a false or multicycle path thru point.
- **In** — Indicates the gate is inside a false or multicycle path combinational intersection cone.
- **Ef** — Indicates the gate is in a false or multicycle path combinational effect cone.
- **--** — Indicates the gate is outside the false or multicycle path region.

The report can also include numbers, which indicate that a gate is part of at least one multicycle path. If there are two numbers, they indicate the minimum and maximum number of cycles. For example, “1,5” means the gate is on a multicycle path with minimum of 1 cycle and maximum of 5 cycles. If there is only one number, it means that the minimum and maximum number of cycles are the same. For example, “2” means the gate is on a 2-cycle path. If there is no number, then it is either on a 1-cycle path (when “--” is shown), or on a false path (when at least one of Fr, To, Th, Ef, In is shown). The following shows how the numbers display in set_gate_report command output:

```
// /I3/reg1    sff
//      D      I  (--)  /cnt[2]
//      SI     I  (--)  /I0/i0/Q
//      SE     I  (--)  /scan_en
//      CLK    I  (--)  /clock
//      Q      O  (In/1,5)  /I3/ix28/S0  /I3/ix30/S0  /I3/ix32/S0
//      QB     O  (--)  /I3/reg3/SI
```

The -Timing_exceptions argument does not reset the previous specified gate report option; consequently, you can use this argument in conjunction with other set_gate_report settings. Refer to “[Example 8](#)” on page 2166.

- **-COnstrain_value {ON | OFF}**

An optional switch and literal pair that specifies that the [report_gates](#) command displays the simulated values that result from all natural tied gates, learned constant value non-scan cells, constrained pins, and constrained cells, as well as the unused flag (UU) which indicates the gate is unreachable from any observation point.

The report_gates command displays four values that are separated by a slash (/). These values are the gate constrained value (-, 0, 1, X, or Z), the gate forbidden values (-, 0, 1, Z, or any combination of 01Z), the fault blockage status (- or B, where B indicates all fault effects of this gate are blocked), and the unused flag (UU).

The displayed values are only available at the end of DRC once the tool successfully leaves Setup Mode. While tie_value reports the status of node values once it comes out of the DRC

checks, the constrain_value tells you what the gate is supposed to act like for future consideration. Refer to [Example 14](#) for an example of this switch.

- **-MAX_Fanouts {number | UNlimited}**

An optional switch paired with either a positive integer or keyword that specifies the maximum of fanout gates to be reported by the [report_gates](#) command. When you specify the Unlimited keyword, report_gates reports all fanout gates without truncation. The default is limited to 99 fanout gates. The tool issues a message if a reported gate drives more than the limit you specified with this command.

- **-CLOCK_domains_in_capture_cycles {ON | OFF}**

An optional switch that enables the report_gates command in the primitive level to report the clock domain of the capture cycles information. The default is off and no clock domain information will be reported. When set to ON, you are allowed to display the launch and capture clocks of a given gate in non-SETUP mode. Both the launch and capture clocks are analyzed considering the constrain values and a clock will not be included when it is blocked by the constrained values.

Note



The clock domain information will not be shown in SETUP mode since the capture cycle constrain values are not analyzed in SETUP mode.

Examples

Example 1

The following example sets the gate report so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the setup system mode):

```
set_system_modeanalysis
set_gate_report error_pattern
report_gates i_1006/o
```

Example 2

The following example illustrates a shift procedure containing two clocks that are pulsed in sequence and the corresponding gate report display when the gate report is set to trace. The data displayed is with respect to the shift procedure.

```
procedure shift =
    force_sci      0;
    measure_sco   0;
    force clk    1 1;
    force clk    0 2;
    force tclk   1 3;
    force tclk   0 4;
    period        6;
end;

report_gates 32
```

```
// /I_15 (32) NOR
//   i0    I (XXXXXX) 16-/I_16/out
//   i1    I (XXXXXX) 17-/I_6/out
//   out   O (XXXXXX) 43-/S2

report_gates tclk

// /TCLK (9) PI
//   TCLK  O (00010) 40-/I_13/clk

report_gates clk

// /CLK (4) PI
//   CLK   O (01000) 20-/I_20/I_225/i0
21-/I_23/I_225/i0
```

Example 3

The following is sample output of the report_gates command for reporting path delay patterns.

```
set_system_mode analysis
create_patterns
set_gate_report pattern_index 0
report_gates /FF1

// /FF1 SLSdffQ0ff
//   CLK   I (L0-L1-L0) /P_CLK1
//   M2SO  I (L0-L0-L0) /M2SOi0/Q
//   D     I (H1-HX-HX) /I2
//   SI2M  I (L0-L0-L0) /SI2Mi0/Q
//   SI    I (0-0-0) /FF2/SO
//   MHN   I (L1-L0-L1) /MHNi0/Q
//   Q     O (0-0-1) /IN2/A
//   SO    O (X-X-X) /P_SO1
```

The Ls and Hs in the gate report refer to clock-off values. These are only displayed for path-delay faults. They represent the clock off value and can be either X, L, H, or Z. The tool reports the clock off state prior to the actual pin value only for pins that are on a clock path.

The clock path value is a value that appears on a gate when pin constraints are considered and when all clocks are at their defined off states. For example, assume your design has an OR gate where one input is connected to primary input pin A. A is constrained to value 1, and the other pin is connected to primary input pin CLK, which is defined as a clock with off-state 0. The clock path values, where Y is the output of the OR gate, would be:

A	H
CLK	L
Y	H

Now, assume A is a regular input pin; that is, it has no pin constraint and is not a clock. The clock path values now are:

A	X
CLK	L
Y	X

Y is in the clock path and if you use “`set_gate_report clock_cone CLK`”, you will see that Y is in the clock cone. Finally, assume that this gate is now an AND gate and A is still a regular input pin. The clock path values are:

A	X
CLK	L
Y	L

Since the clock path depends on both clocks and pin constraints, a pin that is in the clock path does not necessarily have to be in the clock cone.

An S followed by 0, 1, or X appears in patterns generated when you issue the command `set_pattern_type -Multiple_load On`. The S refers to these patterns being scan load operations.

Example 4

The following example shows sample output of the `report_gates` command when the report type is changed to report simulation data using the following `test_setup` procedure.

```

timeplate gen_tp1 =
    force_pi 0;
    measure_po 10;
    pulse clk1 20 10;
    pulse clk2 20 10;
    period 40;
end;

timeplate gen_tp2 =
    force_pi 10;
    measure_po 30;
    pulse clk1 60 10;
    period 100;
end;

procedure test_setup =
    scan_group grp1;
    timeplate gen_tp1;
    // First cycle (cycle 0), one event (force)
    cycle =
        force clk1 0;
        force clk2 0;
        force reset 0;
        force A 0;
        ...
        force E 0;
    end;
    // Second cycle, three events (force, pulse on, pulse off)
    cycle =
        force B 1;
        pulse reset;
        pulse clk2;
    end;
    // Third cycle, three events (force, pulse on, pulse off)
    cycle =
        force A 1;
        pulse clk2;
    end;
    // Fourth cycle, three events (force, pulse on, pulse off)
    cycle =
        timeplate gen_tp2;
        force A 0;
        pulse clk1;
    end;
end;

```

set_gate_level primitive
set_gate_report drc_pattern test_setup
set_system_mode analysis
report_gates /ff20

```
// /ff20 (54) DFF
// "IO" I 15-
// "I1" I 14-
// CLK I 6-/clk2
// D I 7-/A Cycle count starts at 0.
// "OUT" O 24-
// Cycle: 0 1 2 3 Force at cumulative time 40 (1st event of 2nd cycle).
// ----- - 11 111 Pulse off clk2 at cumulative time 70 (3rd event of 2nd cycle).
// 467 801 389 (Time is read vertically; time unit is based on timescale
// defined in test procedure file - 1 ns by default.)
// Time: 0 000 000 000
// ----- - -----
// "IO" 0 000 000 000
// "I1" 0 000 000 000
// CLK 0 010 010 000
// D 0 000 111 000 Clock pulse on & output update are shown simultaneous.
// "OUT" X 000 011 111 (with zero delay)
```

**set_gate_report drc_pattern test_setup -cycle 2 end
report_gates /ff20**

```
// /ff20 (54) DFF
// "IO" I 15-
// "I1" I 14-
// CLK I 6-/clk2
// D I 7-/A
// "OUT" O 24-
//
// Cycle: 2 3
// ----- -
// 11 111
// 801 389
// Time: 000 000
// ----- -
// "IO" 000 000
// "I1" 000 000
// CLK 000 000
// D 111 000
// "OUT" 000 000
```

Example 5

The following examples show sample output of the report_gates command for sequential elements when the report type is set to report simulation data for a particular pattern. For more information about what the tool displays for each cycle, refer to “[Event Simulation for DFFs and Latches](#)” in the *Tessent Scan and ATPG User’s Manual*.

- Mux-DFF—primitive level

The DFF captured 1 (shown in “[]”) at the end of the cycles. This DFF is also the observe element for this scan cell. Therefore, the captured value is also the unload value.

```
set_gate_report pattern 6
set_gate_level primitive
report_gates /re1
```

```
// /re1 (26) MUX
//   TE I (000-000) 19-/scan_en
//   D I (000-000) 2-/d[1]
//   TI I (000-000) 80-/re0/Q
//   "OUT" O (000-000) 81-
// /re1 (81) DFF
//   "S" I (000-000) 20-
//   "R" I (000-000) 21-
//   CP I (000-010) 62-/rh1/CPB
//   "D0" I (000-111) 26-
//   Q O (000-011-[1]) 103-/sout[0]
//   MASTER cell_id=0 chain=chain0 group=group0
invert_data=FFFF
// Reported all gates in instance re1
```

- Mux-DFF—design level

set_gate_report pattern 6
set_gate_level design
report_gates /re1

```
// /re1 FD1SQM1D
//   D I (000-000) /d[1]
//   CP I (000-010) /rh1/CPB
//   TI I (000-000) /re0/Q
//   TE I (000-000) /scan_en
//   Q O (000-011-[1]) /sout[0]
```

- Non-scan DFF—primitive level

set_gate_level primitive
report_gates /rh1/UDP1

```
// /rh1/UDP1 (25) INV
//   "IO" I (011-111) 79-
//   "OUT" O (100-000) 62-
// /rh1/UDP1 (79) TLA
//   "S" I (000-000) 20-
//   "R" I (000-000) 21-
//   "C0" I (010-000) 61-
//   "D0" I (111-111) 38-
//   "OUT" O (011-111) 25-
// Reported all gates in instance rh1/UDP1
```

- Non-scan DFF—design level

set_gate_level design
report_gates /rh1/UDP1

```
// /rh1 LCLKRHDELLD350D
//   CP I (101-111) /inv0/Z
//   LD I (000-000) /ld[0]
//   TE I (000-000) /scan_en
//   CPB O (100-000) /fout1 /inv1/A /rh11/CP
//                                /nonscan_d_b/D /re0/CP /re1/CP
//                                /nonscan_d_b/TI
```

- LSSD scan cell (master latch and slave latch with master_observe)—primitive level

If reporting the master, which is the observed element, the captured value is also the unload value. If reporting the slave, an “Unobs” appears beside the captured value to remind you that this captured value is not the unload value. Notice also the additional message at the end of the report for the slave, telling you that the master (217) is the observation point.

set_gate_level primitive**report_gates 217 218**

```
// /I26/L1 (217) LA
// "S" I (000-000) 26-
// "R" I (000-000) 20-
// CLOCK I (010-000) 164-/clk_regen/I_crinv_c1/I2/OUT
// "D0" I (111-111) 154-
// SCANCLOCK I (000-000) 193-/clk_regen/I_inv_aclk3/I2/OUT
// SCANIN I (111-111) 104-/testblock/msff_output/I12/OUT
// Q O (111-111-[1]) 33-/I26/I130/IN
// MASTER cell_id=1 chain=sc0 group=g1 invert_data=FFFF
// /I26/L2 (218) LA-IL
// "S" I (000-000) 130-
// "R" I (000-000) 20-
// CLOCK I (000-010) 203-/clk_regen/I_crinv_mc2/I2/OUT
// DATA I (111-111) 33-/I26/I130/OUT
// Q O (111-111-[1 Unobs]) 34-/I26/I12/IN
// SLAVE cell_id=1 chain=sc0 group=g1 invert_data=FFFF
// Latch is not the observation point for reported pattern. //
Observation point is at MASTER gate 217.
// Learned latch port types: IL
```

- Masked LSSD scan cell—primitive level

If the cell is masked (unload value = X due to an XX cell constraint added with the add_cell_constraints command, for example), the tool informs you that the cell is not observed for this pattern.

set_gate_level primitive**report_gates 311**

```
// /ix301/slave (311) LA
// "S" I (000-000) 43-
// "R" I (000-000) 206-
// "C0" I (000-010) 116-
// "D0" I (000-000) 59-
// "OUT" O (111-100 [X Unobs]) 58-
// SLAVE cell_id=5 chain=chain2 group=grp_lssd invert_data=TTFT
// Scan cell not observed for this pattern.
```

- LSSD scan cell, master and slave with master_observe—design level

Notice that the tool does not show explanatory text at the design level for “Unobs” cells.

set_gate_level design**report_gates 217 218**

```

//  /I26/L1 (217)  LA
//    "S"    I  (000-000)  26-
//    "R"    I  (000-000)  20-
//    CLOCK  I  (010-000)  164-/clk_regen/I_crinv_c1/I2/OUT
//    "D0"    I  (111-111)  154-
//    SCANCLOCK  I  (000-000)  193-/clk_regen/I_inv_aclk3/I2/OUT
//    SCANIN   I  (111-111)  104-/testblock/msff_output/I12/OUT
//    Q      O  (111-111-[1])  33-/I26/I130/IN
//    MASTER   cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//  /I26/L2 (218)  LA-IL
//    "S"    I  (000-000)  130-
//    "R"    I  (000-000)  20-
//    CLOCK  I  (000-010)  203-/clk_regen/I_crinv_mc2/I2/OUT
//    DATA    I  (111-111)  33-/I26/I130/OUT
//    Q      O  (111-111-[1 Unobs])  34-/I26/I12/IN
//    SLAVE   cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//    Latch is not the observation point for reported pattern. //
Observation point is at MASTER gate 217.
//    Learned latch port types: IL

```

Example 6

The following example sets the gate report so that reporting and display show all the K19-simulated values of each reported gate:

```

set_gate_report drc_pattern k19
// Warning: Data will be accessible after running DRC.

```

The report_environment command would show the following for the gate report:

```

gate report = display K19 sim data, procedure(s): test_setup,
load_unload, shift - all cycles

```

The following is sample output of the report_gates command when the report type is changed prior to DRC to report K19 simulation data for the following load_unload procedure and shift cycles 8 through 10. The example begins in Setup system mode.

set_gate_report

```

timeplate gen_tp1 =
    force_pi 0;
    measure_po 1;
    pulse clk 2 1;
    pulse edt_clock 2 1;
    period 4;
end;
procedure load_unload =
    scan_group grp1;
    timeplate gen_tp1;
    cycle =
        force clk 0;
        force edt_bypass 0;
        force edt_clock 0;
        force edt_update 1;
        force scan_en 1;
        pulse edt_clock;
    end;
    apply shift 11;
end;
procedure shift =
    scan_group grp1;
    timeplate gen_tp1;
    cycle =
        force_sci;
        force edt_update 0;
        measure_sco;
        pulse clk;
        pulse edt_clock;
    end;
end;
set_gate_report drc_pattern k19 load_unload 8 10
// Warning: Data will be accessible after running DRC.
set_system_mode
report_gates /edt_i/edt_decompressor_i/ix159/Y
// /edt_i/edt_decompressor_i/ix159 (142) NXOR
//      A0 I 130-/edt_i/edt_decompressor_i/ix241/Y
//      A1 I 84-/edt_i/edt_decompressor_i/lfsm_lockup_4/Q
//      Y O 154-/edt_i/ix90/A
// Proc: ld_un ~ shi 8 sh 9 sh10
// ----- ~ ----- -----
// Time: i0234 ~ i0123 0123 0123
// ----- ~ ----- -----
// Sim: XXX00 ~ 00000 0001 1110
// Emu: ---- ~ 0---0 ---1 ---0
// No mismatch.
// Monitor: EDT decompressor chain 1 output.

```

In the preceding report, consecutive cycles are listed from left to right, with a “~” indicating a break in the data. A blank space separates each cycle’s data from that of neighboring cycles. The numbers in the row labeled “Time” are the relative times of each event in a cycle as specified in the procedure file. An “i” indicates the start of a new sequence of events. Because the gate is one the tool monitors for the purpose of pinpointing where K19 errors first occur, the report shows the values the tool emulated for the gate and whether there were any mismatches between the emulated and simulated values.

Example 7

The following example sets the gate report so reporting and display use the K22 simulated values for the load_unload procedure of the preceding example and shift cycles 4-8 for the fourth core scan chain.

```
set_gate_report drc_pattern k22 -mask 4 4 8
// Warning: Data will be accessible after running DRC.
```

The report_environment command would show the following for the gate report:

```
gate report = display K22 sim data mask 4, procedure(s) :
    load_unload, shift - range 4 - 8
```

The following is sample output of the report_gates command when the report type is changed prior to DRC. The example begins in Setup system mode.

```
set_gate_report drc_pattern k22 -mask 4 4 8
// Warning: Data will be accessible after running DRC.
set_system_mode

report_gates /edt_i/edt_bypass_logic_i/ix7/Y

// /edt_i/edt_bypass_logic_i/ix7 (216) OR
//      "I0" I 215-
//      B0   I 76-/edt_i/edt_bypass_logic_i/ix5/Y
//      Y     O 256-/edt_channels_out1
//
// Proc: ld_un ~ shi 4 sh 5 sh 6 sh 7 sh 8
// ----- ~ ----- -----
// Time: i 234 ~ i 123 123 123 123 123
// ----- ~ ----- -----
// Sim: 11000 ~ 11100 0011 1111 1100 0011
// Emu: ---- ~ --0-- -1-- -0-- -0-- -1--
// Mism: ~ * * * * *
// Monitor: EDT module channel 1 output.
```

The report format is similar to that of the K19 report. Here too, the gate is one the tool monitors for the purpose of analyzing where K22 errors first occur, so in addition to the simulation results, the report shows the values the tool emulated for the gate and whether there were any mismatches. Notice that the emulated values occur earlier in each cycle than with the K19 DRC. The reason is that the K22 DRC synchronizes emulation comparisons with the measure_sco statements in the shift procedure, whereas the K19 comparisons always occur at the end of each shift cycle.

Note

 If you enter a constraint using a tool command, Tesson TestKompress does not apply it until the end of the test_setup procedure. You may thus see Xs up until the end of test setup; however, the tool does correctly apply the constraint before the start of load_unload or capture.

Example 8

The following example demonstrates the reporting of the timing exception and an internal pattern simulation value, and illustrates using the [report_gates](#) command to trace the transition of the pattern along the timing exception paths:

```
set_gate_report -timing_exceptions on
set_gate_report pattern_index 1
report_gates /ix21/UD1
report_gates /ix21/UD1

// /ix21/UD1 (36) DFF
// "S" I (0-0) (--) 6-
// "R" I (0-1) (--) 20-
// "C0" I (1-0) (--) 10-
// "D0" I (1-1) (To) 32-
// "OUT" O (1-1 [0]) (In/Ef) 9-
// MASTER cell_id=2 chain=chain1 group=grp1 invert_data=FFFF

f
// /ix21/UD1 (9) BUF
// "I0" I (1-1) (Fr/Ef) 36-
// "OUT" O (1-1) (In) 18- 19-

f
// /ix21 (18) BUF
// "I0" I (1-1) (In) 9-
// Q O (1-1) (To) 28-/fdgd/A 29-/ix19/A

f
// /fdgd (28) NAND
// A I (1-1) (To) 18-/ix21/Q
// B I (1-0) (--) 1-/s
// Z O (0-1) (Ef) 37-/y[2]

f
// /y[2] (37) PO
// "I0" I (0-1) (Ef) 28-/fdgd/Z
// y[2] O (0-1) (Ef)
```

Example 9

The following example reports the static propagation time of gate */U1*:

```
set_gate_report delay_data static_propagation
report_gates /U1

// /U1 (24) XOR
// AO I (5.4000, 5.5000)/(5.0000, 5.1000) 1-/PI1
// A1 I (2.4000, 2.4000)/(2.0000, 2.0000) 16-/U11/Y
// Y O (2.4000, 2.0000)/(2.4000, 2.0000) 25-/U12/Y
```

The following example reports the static slack test of /U2:

```
set_gate_report DELay_data STATIC_Slack
report_gates /U2

// /U2 sff
// D I (5.0000/2.0000) /U21/Y
// SI I (12.0000/12.0000) /U22/Q
// SE I (NA/NA) /SE
// CLK I (8.0000/8.0000) /CLK
// RB I (STAT) /U23/Y
// Q O (2.0000/3.0000) /U24/SI /U25/A0 /U26/A1
```

The following example reports the local delay data for /U3. The output in bold font highlights the timing information used by ATPG. Note, to report local delay data, you must set the gate level to primitive before executing the report_gates command as shown here:

```
set_gate_level primitive
set_gate_report delay local_delay_data
report_gates U3
```

```

//  /U3  (50)  BUF
//    "IO"    I  (-)  101-
//    Q      O  (-)  85-/U31/SI
//  /U3  (51)  INV
//    "IO"    I  (-)  101-
//    QB     O  (-)
//  /U3  (84)  MUX
//    SE     I  (-)  56-
//    D      I  (-)  78-/U32/Y
//    SI     I  (-)  48-/U34/Q
//    "OUT"   O  (-)  101-
//  /U3  (101) DFF
//    "S"    I  (-)  69-
//    "R"    I  (-)  17-
//    CLK   I  (-)  54-
//    "D0"   I  (-)  84-
//    "OUT"   O  (-)  50-  51-
//
// Sequential IOPATH delay starts through the fanin 3 of /U3/ (101):
//   1 condition delay(s) to /U3/Q (50)
//   condition delay: (1.0000, 2.0000)
//   condition variable list:
//     V[0]=54
//   condition expression in post fix format: (V[0]) posedge
// TimingCheck for data port at the fanin 2 of /U3/ (84):
//   SETUP - clock port at the fanin 3 of /U3/ (101)
//   condition delay: (0.7000, 0.7000)
//   condition variable list:
//     V[0]=54
//   condition expression in post fix format: (V[0]) posedge
// HOLD - clock port at the fanin 3 of /U3/ (101)
//   condition delay: (0.2000, 0.2000)
//   condition variable list:
//     V[0]=54
//   condition expression in post fix format: (V[0]) posedge
//
//   MASTER  cell_id=1  chain=c1  group=g1  invert_data=FFFF
// Reported all gates in instance U3

```

Example 10

The following example reports gates with MCP (multicycle path) information, listing the minimum and maximum number of cycles. This example assumes that the following MCP information has been set from an SDC (Synopsys Design Constraint) file:

```

set_multicycle_path -through { get_pin [b1/I] } -setup 2

set_gate_report -timing_exceptions on
report_gates /b1/ci2b cg2/TEN /cg2/i03

```

```
// /b1/ci2b (47) INV
//   A   I  (To/Th/1,2)  40-/b1/ci2a/Y
//   Y   O  (Ef/1,2)    49-/cg2/CLKEN
// /cg2 (49) NOR
//   CLKEN I  (Ef/1,2)  47-/b1/ci2b/Y
//   TEN   I  (--)      3-/se
//   "OUT" O  (Ef/1,2)  51-
// /cg2/i03 (51) TLA
//   "S"   I  (--)      9-
//   "R"   I  (--)      6-
//   "C0"  I  (--)      44-
//   "D0"  I  (Ef/1,2)  49-
//   "OUT" O  (Ef/1,2)  11-
```

When the minimum and maximum number of cycles are the same and larger than 1, the command reports only one number. Single cycle paths and false paths do not report any number.

Example 11

The following example specifies that a subsequent report_gates command display the simulated values from the current simulation context:

```
> set_current_simulation_context stable_after_setup
> set_gate_report simulation_context
> report_gates {I1 I2}

/I1 primary_input
  I1   O  (0)  /AND2/A0
/I2 primary_input
  I2   O  (1)  /AND2/A1
```

Example 12

The following example enables the report_gates command to show the simulation values for the first F20 violation:

```
set_gate_report drc_pattern F20-1
```

Example 13

The following example shows how to display the clock domain information of a combinational gate and a sequential gate. The -clock_domains_in_capture_clocks switch, when combined with the pattern_index switch, shows both clock domain information as well as the pattern simulation values.

```
set_system_mode analysis
create_patterns
set_gate_level primitive
set_gate_report pattern_index 0
set_gate_report -clock_domains_in_capture_cycles on
report_gate /ff1_3/D
```

set_gate_report

```
//  /ff1_3 (30) MUX
//    "SE"   I (000) 9-/scan_en
//    "D"    I (111) 11-/pi2
//    SI     I (000) 128-/latch3/Q
//    "OUT"   O (111) 129-
// Launch clock:      '/clk1' (1)
// Capture clock:    '/clk1' (1)
```

report_gate /dff10/QB

```
//  /dff10 (109) BUF
//    "I0"   I (111) 62-
//    "QB"   O (111) 121-/chain5_1/ff5_1/D 122-/or/A1
// Launch clock:      '/clk2' (2)
// Capture clock:    '/clk3' (3),'/reset' (18)
```

Example 14

This example demonstrates the effect of the `-constrain_value` switch on the `report_gates` command. When you have specified the “`set_gate_report -constrain_value on`”, the `report_gates` command will show when a gate is unreachable with the unused flag (UU). The report for the INV gate in /sdff2/lat pin QN shows (X/01/-/UU):

- X — Gate constrained value. In this case the gate is constrained to X.
- 01 — Gate forbidden values are 0 and 1.
- - (hyphen) — Fault blockage status. In this case - (hyphen) means no fault blockage.
- UU — Unused flag (indicating gate is unreachable from any observation point).

set_gate_report -constrain_value on
report_gates sdff2/lat

```
//  /sdff2/lat (18) BUF
//    "I0"   I (X/01/-) 34-
//    Q     O (X/01/-) 36-/so
//  /sdff2/lat (19) INV
//    "I0"   I (X/01/-) 34-
//    QN    O (X/01/-/UU)
//  /sdff2/lat (34) TLA
//    "S"    I (0/1/-) 9-
//    "R"    I (0/1/-) 8-
//    CLK   I (-/-/B) 26-/sdff2/u0/OUT
//    D     I (-/-/-) 29-/sdff2/u1/OUT
//    "OUT"  O (X/01/-) 18- 19-
// Reported all gates in instance 'sdff2/lat'
```

Example 15

This procedure file defines four events in the shift procedure (`force_pi` is set to occur at 5 ns rather than at the beginning of the cycle).

```

set time scale 1.000000 ns ;
set strobe_window time 1 ;
timeplate gen_tp1 =
    force_pi 5 ;
    measure_po 10 ;
    pulse CLOCK 20 10;
    period 40 ;
end;
procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        measure_sco ;
        pulse CLOCK ;
    end;
end;
procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force scan_en 1 ;
        force CLOCK 0 ;
    end ;
    apply shift 3;
end;
procedure test_setup =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force IN1 0 ;
    end;
end;

```

The report_gates command reports four values, one for each event in the cycle.

```

set_gate_level design
set_gate_report drc shift
report_gates .reg1

// /reg1 dfsc
// CK I (0010) /CLOCK
// D0 I (XXXX) /gate1/OUT
// D1 I (SSSS) /reg2/QBAR
// SC I (1111) /scan_en
// Q O (XXXX) /gate2/B
// QBAR O (SSSS) /scan_out1

```

Related Topics

- [report_gates](#)
- [report_display_instances](#)
- [set_gate_level](#)

set_gzip_options

Context: unspecified, all contexts

Mode: all modes

Specifies GNU gzip options to use with the GNU gzip command.

Usage

```
set_gzip_options [-Path gzip_path] [-Fast | -Best | -digit]
```

Description

Specifies GNU gzip options to use with the GNU gzip command.

The set_gzip_options command specifies GNU gzip options the tool will use when compressing or decompressing files using the GNU gzip command. When file compression handling is enabled, the tool uses GNU gzip when processing files having a .gz or .Z extension.

The default speed and amount of compression corresponds to the gzip -4 option.

Arguments

- **-Path *gzip_path***

An optional literal that specifies the full path *gzip_path* to a gzip executable file. You need to provide this pathname only if gzip is not in your normal Linux search path. If you have specified a pathname with this option, you can restore the default behavior of using your Linux path to find gzip by issuing the command with the -Path gzip option.

The following switches all control the speed of compression:

- **-Fast**

An optional switch that specifies to use the fastest compression method, which yields the least compression. This corresponds to the gzip -1 option.

- **-Best**

An optional switch that specifies to use the slowest compression method, which yields the most compression. This corresponds to the gzip -9 option.

- **-*digit***

An optional switch that specifies an integer from 1 to 9 that the tool passes to gzip to control the rate of compression. You obtain the least amount of compression with -1, the greatest amount of compression with -9.

Examples

The following example ensures file compression is enabled, sets gzip compression to the fastest method, then saves a file using the .gz file naming extension in order to activate gzip file compression handling:

```
set_gzip_options -fast
write_patterns fault.pat.gz
```

Related Topics

[write_patterns](#)

set_icl_extraction_options

Context: all contexts

Mode: all modes

Customizes the behavior of ICL extraction.

Usage

```
set_icl_extraction_options [-extract_learned_muxes {ON | OFF}]  
                         [-extract_library_muxes {ON | OFF}]
```

Description

Customizes the behavior of ICL extraction.

For more information, see “[ICL Extraction](#)” in the *Tessent IJTAG User’s Manual*.

Arguments

- **-extract_learned_muxes {ON | OFF}**

An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from structures in the design that have been identified as muxes by the static learning process. The default for this switch is ON.

- **-extract_library_muxes {ON | OFF}**

An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from muxes that are described in the Tessent Cell library. The default for this switch is ON.

Examples

The following command switches off the recognition of multiplexer library cells in ICL extraction:

```
set_icl_extraction_options -extract_library_muxes off
```

Related Topics

[get_icl_extraction_options](#)

[report_icl_extraction_options](#)

[set_static_learning](#)

set_icl_scan_interface_ports

Context: dft, patterns -ijtag

Mode: setup, insertion

Sets the names of the ports in the specified scan interface previously added by add_icl_scan_interfaces that ICL extraction will create in the new ICL top module.

Usage

```
set_icl_scan_interface_ports -name name -ports port_objects  
[-allowed_to_be_tied_externally | -allowed_no_source_externally]
```

Description

Sets the names and connection rules of the ports in the specified scan interface previously added by add_icl_scan_interfaces that ICL extraction will create in the new ICL top module.

Arguments

- **-name *name***

A required switch and value pair that specifies the name of scan interface for which the list of ports will be created in the new ICL top module.

- **-ports *port_objects***

A required switch and value pair that specifies the names of the ports that will be associated with the scan interface in the new ICL top module.

- **-allowed_to_be_tied_externally**

An optional switch that instructs the tool to set appropriate connection rules for the ports of the specified scan interface. Depending on the function of the port and the type of the scan interface, the following connection rules are chosen:

ScanInPort: allowed_tied

ScanOutPort: allowed_no_destination

Controls (client/client-TAP ScanInterface): allowed_tied_low

Controls (host/host-TAP ScanInterface): allowed_no_destination

By using this switch, you can request that a ScanInterface be unused at the next level up without setting that as default.

If the -allowed_to_be_tied_externally and the -allowed_no_source_externally switches are not specified, the ports in the extracted ICL inherit their connection_rule_option attributes from the ports they traced to or from. In the case of an output port with a single driver, the inheritance is one to one. In the case of an input port with potentially more than one fanout, the most restrictive value of all fanouts is used. For example, if an input port fans out to two ports where one has connection_rule_option = allowed_tied and the other allowed_no_source, the allowed_tied value will be used as it is the more restrictive. If any

port in the fanout does not define the [icl_port](#) connection_rule_option attribute, the created port will not have it either.

- -allowed_no_source_externally

An optional switch that instructs the tool to set appropriate connection rules for the ports of the specified scan interface. Depending on the function of the port and the type of the scan interface, the following connection rules are chosen:

ScanInPort: allowed_no_source

ScanOutPort: allowed_no_destination

Controls (client/client-TAP ScanInterface): allowed_no_source

Controls (host/host-TAP ScanInterface): allowed_no_destination

By using this switch, you can request that a ScanInterface be unused at the next level up without setting that as default.

If the -allowed_to_be_tied_externally and the -allowed_no_source_externally switches are not specified, the ports in the extracted ICL inherit their connection_rule_option attributes from the ports they traced to or from. In the case of an output port with a single driver, the inheritance is one to one. In the case of an input port with potentially more than one fanout, the most restrictive value of all fanouts is used. For example, if an input port fans out to two ports where one has connection_rule_option = allowed_tied and the other allowed_no_source, the allowed_tied value will be used as it is the more restrictive. If any port in the fanout does not define the connection_rule_option attribute, the created port will not have it either.

Examples

The following example creates the ports tck, tdi, tdo, tms and trst for the ScanInterface TAP, and the ports tdo, tdrEn1, fromTdr1, ce, se and ue for the ScanInterface Internal in the new ICL top module:

```
set_current_design tap1
add_icl_scan_interfaces {TAP Internal}
set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}
set_icl_scan_interface_ports -name Internal -ports {tdrEn1 fromTdr1 ce se ue}
```

The resulting ICL model will look like this:

```
Module tap1 {  
  
    TCKPort      tck;  
    ScanInPort   tdi;  
    ScanOutPort  tdo      {  Source IRMux;  }  
    TMSPort      tms;  
    TRSTPort     trst;  
    ToSelectPort tdrEn1   {  Source sel1;  }  
    ScanInPort   fromTdr1;  
    ToCaptureEnPort ce;  
    ToShiftEnPort se;  
    ToUpdateEnPort ue;  
  
    ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}  
    ScanInterface Internal {Port tdrEn1; Port fromTdr1; Port ce; Port se;  
    Port ue;}  
}
```

Related Topics

[add_icl_scan_interfaces](#)
[delete_icl_scan_interfaces](#)
[get_icl_scan_interface_list](#)
[get_icl_scan_interface_port_list](#)

set_iddq_checks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.

Usage

```
set_iddq_checks [-NONE | {[-Int_float] [-Bus] [-WEAKBus] [-Pull] [-WIre]  
[-VOLTGain] [-VOLTLoss] [-Clock] [-WRite] [-Read]}  
[-ATpg] [-NO_ATpg] [-WArning | -ERror] [-LOOP_oscillation {OFF | REPort | REject}]
```

Description

These restrictions only apply during the actual time of the IDDQ measurement; they are ignored at other times during a pattern.

To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

```
set_flattener_rule_handling fn1 error  
set_flattener_rule_handling fn3 error
```

When you use the tool to create Iddq patterns, a violation of any of the Iddq Check options you select causes the tool to reject an offending pattern.

During simulation, whenever violations of the restrictions occur, a message displays identifying the gate associated with the violation and the number of patterns in which the violations occurred. The handling of the violation can be either a warning or error.

When you select -Error, simulation terminates at the first occurrence of a violation. You can use the report_gates command to inspect the simulation values of all gates for patterns that violate the restrictions by first using the set_gate_report command with the Error_pattern option.

The [report_environment](#) command reports the current set_iddq_check settings.

For more information on IDDQ, see “[IDDQ Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **-NONE**

An optional switch that specifies to not perform any checks. This is the default.

- **-Int_float** — An optional switch that specifies to not allow a floating (Z) value on any ZVAL primitive gate that drives any Boolean logic. A ZVAL gate is placed between a source (Z producer) and a sink (nonZ consumer).

A Z producer is any primitive that has a Z output. Z producers include top-level inputs, bidi/inouts, tristate drivers, and FETs/switches. A nonZ consumer is any gate that treats Z the same as X for all inputs. NonZ consumers include any Boolean gate. You can see the primitive truth tables for all DFT-supported primitives, in section “[Supported Primitives](#)” of the *Tessent Cell Library Manual*.

Note

 A ZVAL gate is not placed between a Z producer and a PO or the PO half of a bidi because both of these have the ability to output a Z value. However, a ZVAL gate is placed between the PI half of the bidi and any internal _buf as typifies the input path of a bidi/inout IO pad. For those pads, if the output stage is disabled (Z out), and the bidi/inout PAD is driven externally by Z from the tester, and the IO pad has no modeled pullup/down, the ZVAL in front of the _buf input path can have a Z on its inputs; it fails the **-int_float** check.

These “floating” checks only occur at buses. To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

```
set_flattener_rule_handling fn1 error  
set_flattener_rule_handling fn3 error
```

- **-Bus**
An optional switch that specifies to not allow contention conditions on bus gates.
- **-WEAKBus**
An optional switch that prevents contention conditions on weak-bus gates.
- **-Pull**
An optional switch that prevents contention conditions on pull gates.
- **-WIre**
An optional switch that specifies that all inputs of a wire gate must be set to the same value. There should be no contention on wires during IDDQ measurements because contention can raise the IDDQ current.
- **-VOLTGain**
An optional switch that specifies that a PMOS transistor must not be at a logic zero unless a zholt0 or zholt bus_keeper DFT library attribute is between the PMOS transistor and the BUS root. A compromised (Vss+) logic zero is pulled down to the Vss rail (zero volts).

- **-VOLTLoss**

An optional switch that specifies that a NMOS transistor must not pass a logic one unless a zhold1 or zhold bus_keeper DFT library attribute is between the NMOS transistor and the BUS root. A compromised (Vdd-) logic one is pulled up to the Vdd rail (Vdd volts).

- **-Clock**

An optional switch that specifies to not allow clock pins to be on during the IDDQ measure.

Note

 The advantage of the -Clock switch, during pattern generation, is that it prevents creation of Clock PO style patterns, one of the causes of multiple timeplates. The switch also ensures that when a clock is turned on, it is pulsed at the normal time.

- **-WRite**

An optional switch that specifies to not allow write control pins to be on during the IDDQ measure.

- **-RRead**

An optional switch that specifies to not allow read control pins to be on during the IDDQ measure.

- **-ATpg**

An optional switch that specifies to satisfy any IDDQ restrictions set using the [add_iddq_exceptions](#) command. This is the default.

- **-NO_ATpg**

An optional switch that specifies to not enforce IDDQ restrictions set using the [add_iddq_exceptions](#) command. Patterns that do not satisfy restrictions are rejected by the fault simulator.

- **-WArning**

An optional switch that treats violations of IDDQ checks as warnings. This is the default.

- **-ERror**

An optional switch that treats violations of IDDQ checks as errors and stops the simulation process immediately.

- **-LOOP_oscillation {OFF|REPort | REject}**

An optional switch and keyword pair that checks for oscillating loops when the fault type is IDDQ. The “Report” option reports on oscillating simulation values at specific gates. The “Reject” option automatically rejects any patterns that cause oscillation and issues a warning. By default, the tool does not check for oscillating loops.

Examples

Example 1

The following example creates IDDQ patterns while checking that internal buses are not floating during an IDDQ measure and requires the deterministic ATPG to ensure that this is true:

```
set_system_mode analysis
set_fault_type iddq
set_iddq_checks -int_float
create_patterns
```

Example 2

This example uses the report_environment command to report the set_iddq_checks settings:

```
// command: set_fault_type iddq
// command: set_iddq_checks -Bus -Int_float -warning
// command: report_environment
```

```

abort limit = 30
atpg compression = OFF
atpg limits = none
bist initialization = X
bus simulation method = global
capture clock = CLK
checkpoint = OFF
clock restriction = domain_clocks (same-edge interaction)
clock_off simulation = OFF (optimal)
contention check = ON, mode = bus, handling = warning
DRC transient detection = ON -NOVerbose
fails report = OFF
fault mode = uncollapsed
fault type = IDDQ
gate level = design
gate report = normal
iddq checks = bus int_float , handling = warning -atpg
iddq strobe = label
learn reporting = OFF
logfile handling = OFF
net dominance = WIRE
net resolution = WIRE
observation point = master
pattern classification = ON
pattern source = internal
pattern type = sequential_depth = 0, multiple_load = OFF,
ram_sequential = OFF, clock_po = ON
possible credit = 50%
pulse generators = OFF
RAM initialization = uninitialized
RAM test mode = static_pass_thru
random atpg = ON
random clocks = none
random patterns = 1024
screen display = ON
shadow checking = ON
skew load = OFF
split capture cycle = OFF (optimal)
stability check = ON
system mode = atpg
TLA loop handling = ON
trace report = OFF
Z handling = int=X ext=X
Zhold behavior = ON

```

Related Topics

[add_iddq_exceptions](#)
[analyze_restrictions](#)
[delete_iddq_exceptions](#)
[report_gates](#)
[report_iddq_exceptions](#)

[**set_fault_type**](#)
[**set_gate_report**](#)

set_ijtag_instance_options

Context: dft patterns

Mode: setup analysis insertionset_icl_scan

Sets options on instances for which there is a matched ICL module.

Usage

```
set_jtag_instance_options ijtag_instance_spec [-use_in_dft_specification off | auto]  
[-connect_data_signals_already_connected_to_ports on | off | auto ]
```

Description

Sets options on instances for which there is a matched ICL module.

Arguments

- *ijtag_instance_spec*

A required literal that is a Tcl list of names of one or more Ijtag instances, or a collection containing one or many Ijtag instances as reported by the [get_instances](#) or [get_ijtag_instances](#) commands.

The instance objects must have an ICL module matched to them.

- -use_in_dft_specification off | auto

An option value pair that defines the value of the `-use_in_dft_specification` option. When set to `auto`, the instances are included in the `DftSpecification` created by the `create_dft_specification` command when it is seen to not already be connected to the network. Specifying an `off` value bypasses the analysis and forces the inclusion or exclusion of the instances. See the description of the `create_dft_specification` command for more details about the usage of this option.

- -connect_data_signals_already_connected_to_ports on | off | auto

An option value pair that specifies how DataInPorts and DataOutPorts on the specified instances are handled when they are seen to be already connected to PI/PO ports of the current design. When set to `on`, the `create_dft_specification` command adds a TDR to control the DataInPorts and observe the DataOutPorts even if they are already connected to PI/PO ports of the current design. When set to `off`, no TDR is added to control the DataInPorts and observe the DataOutPorts when they are already connected to PI/PO ports of the current design. The assumption is that they are already observable or controllable by the top level ports. When set to `auto`, the value will be taken from the [DefaultsSpecification\(policy\)/DftSpecification/IjtagNetwork/connect_data_signals_already_connected_to_ports](#) property. You set this option to `on` when you want to add observability and controllability through ijtag using an internal register even if they are already observable/controllable by ports of the current design.

Examples

The following example sets the `-use_in_dft_specification` option to off on all Ijtag instances below the instance blockA.

```
> set_ijtag_instance_options [get_ijtag_instances -below_instance blockA] \
   -use_in_dft_specification off
```

Related Topics

[get_memory_instances](#)
[report_ijtag_instances](#)
[get_ijtag_instance_option](#)

set_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Enables you to configure different aspects of ijtag retargeting.

Usage

```
set_ijtag_retargeting_options [-test_setup_network_end_state {keep | reset}]  
    [-test_setup_ireset {off | first_icall | when_icl_present}]  
    [-iapply_target_annotations {off | dense | full}]  
    [-max_operations_per_iapply integer]  
    [-compare_constant_capture_values {ON | OFF}]  
    [-compare_constant_data_out_ports {ON | OFF}]  
    [-merge_irunloop_only {on | off}]  
    [-inject_cycles inject_cycles_list]  
    [-tck_ratio tck_ratio {auto | 1 | 2 | 4 | 8 | 16 | ...}]  
    [-annotation_parameter_values parameters]
```

Description

Enables you to configure different aspects of ijtag retargeting.

Arguments

- `-test_setup_network_end_state {keep | reset}`

Note

 This switch only applies to ijtag retargeting when an iCall is used in test_setup (see the [set_test_setup_icall](#) command for more details).

An optional switch and literal pair that specifies to switch off the reset and preserve the state of the ICL network after the last test_setup iCall. The default is to do the reset. This reset only affects the configuration of the scan paths and the selection of the scan interfaces. The reset does not trigger any activity on ports of type ResetPort or TRSTPort.

If you have a test mode where you are applying the load_unload data through the IJTAG network and do not want to reset the state of the ICL scan network after the last test_setup iCall, use the “set_ijtag_retargeting_options -test_setup_network_end_state keep” command to turn off the reset of the ICL network for those designs.

- `-test_setup_ireset { off | first_icall | when_icl_present }`

An optional switch and literal pair that specifies if and when an iReset will be automatically added to the beginning of the test setup procedure. Choose one of the following options:

- off

No iReset will automatically be added to the test_setup procedure

- first_icall

An iReset will be automatically added when the first iCall or iMerge is encountered in the test_setup procedure and no user-specified iReset precedes this iCall or iMerge.

- when_icl_present

An iReset will be automatically added if there is ICL for the current design. This is the default setting. It triggers the creation of an appropriate reset sequence at the ICL ports even if the test_setup procedure does not contain any iCall or iMerge statements.

- **-iapply_target_annotations {off | dense | full}**

An optional switch and literal pair that specifies the style of the annotations that show the processed iWrite and iRead targets of an iApply. Choose one of the following options:

- off

Annotations are not created.

- dense

No bit-blasting of registers and ports, binary values with many bits are abbreviated. This is the default.

- full

Registers and ports are bit-blasted, one bit per line, with no abbreviations.

- **-max_operations_per_iapply *integer***

An optional switch and integer pair that specifies the upper limit of scan loads and/or parallel I/O operations that are used to retarget one single iApply. Allowed values are in the range from 4 to 65536, the default is 64. In very complex ICL networks, it might be necessary to increase this limit in order to provide more degrees of freedom to the retargeting engine. But usually the predefined limit will only be reached if it is actually impossible to resolve the current iApply, for example, because of conflicting iWrite commands, a deadlock in the ICL network, and so forth. Please note that increasing this limit can result in very long runtimes of the iApply processing.

- **-compare_constant_capture_values {ON | OFF}**

An optional switch and literal pair that turns on or off the automatic expected value generation for the scan registers.

If turned on, the patterns will not include any extra scan loads. Instead, when a bit happens to be scan out in a given scan load and the value of this bit is known from the CaptureSource description in ICL, a compare value is automatically inferred for that bit at no extra cost. This feature is very useful for diagnosing issues with the network and should be kept on unless the ICL is incorrect.

If the test_setup procedure contains PDL commands, then the inferred compare values (when they exist) are part of the validation of the test_setup procedure (see [E14](#)), and they

are part of the pattern files that contain the vectors for the test setup. If the PDL is processed in an open pattern set in the context “patterns -ijtag”, then the inferred compare values (when they exist) are part of the generated pattern sets. Therefore they will also be part of the pattern files and the retargeted PDL files that are created from those pattern sets when using the write_patterns command. If the inferred compare values are unwanted, the related option of the set_ijtag_retargeting_options command must be turned off before the PDL retargeting happens and during retargeting run. It is not possible to suppress the inferred comparison in existing pattern files or existing retargeted PDL files, if the “-compare_constant_capture_values” or “-compare_constant_data_out_ports” switches were on during the PDL retargeting phase.

- **-compare_constant_data_out_ports {ON | Off}**

An optional switch and literal pair that turns on or off the expected value generation for the primary DataOut ports. If turned on, the patterns will not include any extra vectors. An automatic compare value is inferred on top-level DataOutPorts if its value is known from its described source in ICL. This feature is useful for confirming the correctness of the ICL but is not turned on by default to avoid increasing the number of used ports in a pattern file. For example, if you are exporting the pattern in SVF format, you typically would not want to have to compare output pins other than TDO.

The usage of the -compare_constant_data_out_ports switch impacts the validation of the test_setup procedure, the pattern files and the retargeted PDL. For more information, see the description of the -compare_constant_capture_values switch above.

- **-merge_irunloop_only {on | off}**

An optional switch and literal pair that specifies if iApply statements of concurrent iCalls are allowed to be merged during the processing of an iMerge block. Merging iApply statements can be very complex and the retargeter may spend a lot of time on this task, while the actual benefit (reduction of test time) is negligible in case of tests in which the test time is mainly determined by iRunLoop statements. In such situations, this switch can be set to “on” in order to suppress the merging of iApply statements and to speed up the iMerge computation.

- **-inject_cycles *inject_cycles_list***

An optional switch and string pair that allows the insertion of “dead” cycles (cycles without TCK activity) while the TAP controller is in a certain state. The number of injected cycles is programmable and depends on the TAP state that has just been entered. The *inject_cycles_list* can be a single number or a Tcl list with an even number of arguments, each comprising a TAP state and the related number of injected cycles in alternating order.

You can choose one or more of the following TAP states:

Table 6-10. Allowed TAP States

DRCAPTURE	IRCAPTURE
DRSHIFT	IRSHIFT
DREXIT1	IREDIT1

Table 6-10. Allowed TAP States (cont.)

DRPAUSE	IRPAUSE
DREXIT2	IREXIT2
DRUPDATE	IRUPDATE
IDLE	RESET

The names of the TAP states are consistent with SVF.

The cycles are injected immediately after the TAP state machine has entered the specified state, when the TCK is back to its off-state again. This changes the behavior of the TAP state machine such that the related control signals of a specified TAP state are active for a longer time than they would be usually. Note that the TRST control signal of the TAP is inactive during injected cycles in the RESET state.

The specification of the dead cycles refers to the duration of a TCK cycle, not to the duration of a tester cycle.

The `-inject_cycles` option only affects pattern files in which dead cycles during TAP state transitions can be represented. The output of “`write_patterns -svf`” and “`write_patterns -pdl`” is not changed.

The effect on the pattern files will be the same as using an “`iRunLoop -time`” command. This avoids the TCK pulse. iRunLoops will be inserted at the appropriate places in the unrolled version of the pattern sets. These loops will also be visible by the pattern set introspection command “`get_pattern_set_data`”.

- `-tck_ratio {auto | 1 | 2 | 4 | 8 | 16 | ...}`

An optional switch and literal pair that specifies the TCK ratio for the use of PDL in `test_setup` and `test_end` procedures and for the PDL retargeting in the context patterns `-ijtag`. The default is 4.

In `test_setup` and `test_end` procedures, the waveforms of the TCK ports are either created appropriately (if they have not been specified before) or they are checked with respect to compatibility with the `-tck_ratio` specification.

In the context patterns `-ijtag` the TCK ratio specification of the command `set_ijtag_retargeting_options` becomes effective only if the TCK ratio specification in the command `open_pattern_set` is omitted (“auto” results in a TCK ratio of 1 in this case).

- `-annotation_parameter_values parameters`

An optional switch and list of values to configure the behavior of notes, annotations and comments in Tesson IJTAG. The list must consist of an even number of elements, keys and

values in alternating order. The following table shows an overview of the various configuration possibilities, default values are underlined:

Table 6-11. Annotation Parameter Values

key	possible values	explanation
user	<u>on</u> off	iNote specifications
iapply_targets	off <u>dense</u> full	Annotations about iApply targets, the same as set_ijtag_retargeting_options -iapply_target_annotations
scan_path_registers	<u>on</u> off	Annotations about the scan registers of a scan path
scan_load_unload_value	<u>on</u> off	Annotations about the values of a scan load/unload
tap_states	<u>on</u> off	Annotations about the TAP state transitions in pattern files in which each scan load is unrolled into a series of test vectors. The vectors that correspond to a TAP state transition are annotated with the destination TAP state
network_end_state	<u>on</u> off	Comment indicating that the following test vectors establish the specified network end state, see close_pattern_set -network_end_state set_ijtag_retargeting_options -test_setup_network_end_state
pragma_annotation	<u>on</u> off	The output of “tessonPragma annotation” in the pattern files, that is, the annotation with the original iRead targets, per scan load
pragma_bit_annotation	<u>on</u> off	The output of “tessonPragma bit_annotation” in the pattern files, that is, the annotation with the original iRead targets, per cycle
pragma_variable	<u>on</u> off	The output of “tessonPragma variable” in the pattern files
pragma_bit_variable	<u>on</u> off	The output of “tessonPragma bit_variable” in the pattern files
pragma_icl_checksum	<u>on</u> off	The output of “tessonPragma icl_checksum” in the pattern files
pragma_clock	<u>on</u> off	The output of “tessonPragma clock” in the pattern files
pragma_pattern_set	<u>on</u> off	The output of “tessonPragma pattern_set” in the pattern files
pragma_tester_period	<u>on</u> off	The output of “tessonPragma tester_period” in the pattern files

Table 6-11. Annotation Parameter Values (cont.)

key	possible values	explanation
pragma_expect_z	<u>on</u> off	The output of “tessent_pragma expect_z” in SVF files
svf_command_number	<u>on</u> off	A comment with the command number in SVF files
icl_network_verify_activate	<u>on</u> off	Comments indicating that the following test vectors establish a scan path in the ICLNetworkVerify patterns
icl_network_verify_flush	<u>on</u> off	Comments indicating that the following test vectors are flush patterns, that is, patterns that are loaded and unloaded without performing an “update” and a “capture” operation in between

Examples

Example 1

The following example loads the ICL and turns off the reset of the ICL network after the last test_setup PDL command.

```
SETUP> read_icl ..\data\chip.icl
SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep
```

Example 2

The following example uses the -inject_cycles option to extend the time in which the TAP rests in “capture-DR” state or “capture-IR” state for the duration of three TCK cycles.

```
set_ijtag_retargeting_options -inject_cycles {DRCAPTURE 3 IRCAPTURE 3}
```

The resulting pattern set will contain three extra cycles without TCK pulse after each vector that brings the state machine to Capture-DR, and three extra cycles without TCK pulse after each vector that brings the state machine to Capture-IR.

To inject five dead cycles in all of the twelve allowed TAP states, omit the state when you issue the -inject_cycles command.

```
set_ijtag_retargeting_options -inject_cycles 5
```

If you want to reset everything, you can do so as follows:

```
set_ijtag_retargeting_options -inject_cycles 0
```

Related Topics

[get_ijtag_retargeting_options](#)

[iRead](#)

[report_ijtag_retargeting_options](#)

iWrite

iApply

set_insertion_options

Context: unspecified, all contexts

Mode: all modes

Specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.

Usage

```
set_insertion_options [-auto_uniquify_edited_modules {on | off | when_needed }]
[-edited_module_prefix edited_module_prefix]
[-net_uniquification_suffix net_uniquification_suffix]
[-module_uniquification_suffix module_uniquification_suffix]
[-instance_uniquification_suffix instance_uniquification_suffix]
[-allow_assigns {on | off}]
[-inserted_object_comment_label label]
[-treat_common_ancestor_as_unique {on | off}]
[-preserve_paths_for_unrolled_v2001_loops {on | off}]
[-vhdl_port_pin_net_type {bit | std_logic | std_ulogic}]
[-open_input_pins {tie0 | filler_nets}]
[-remove_inactive_sections_in_unrolled_generate_loops {on | off}]
[-trace_across_module_boundaries_to_find_functional_drivers {on | off}]
```

Description

Specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.

Arguments

- `-auto_uniquify_edited_modules {on | off | when_needed }`

An optional switch and literal pair that enables Tesson Shell to uniquify any module that is edited, either unconditionally or only when needed. This will only occur if a module has more than one instance in the current design.

When set to on, a module will be uniquified on the first edit. For example if all instances are edited, the number of unique copies created will equal the number of instances, in addition to the original.

When set to when_needed, a copy of a module is created and uniquified for each parameterized view rather than for each instance. For example if all instances are edited and they are instantiated over two parameterized views, the number of unique copies created will equal two, in addition to the original.

Note



The current design is the design that was specified using the `set_current_design` command.

During uniquification, the new module name is constructed by prefixing the original module name with the specified or default module_uniquification_suffix. The instantiation call of the module in its parent module is automatically changed to use the new module name. If the parent module is also non-unique, it is uniquified until a unique parent module is reached.

- **-edited_module_prefix *edited_module_prefix***

An optional switch and string pair that specifies a prefix to add to every edited module. A valid prefix contains numbers, letters, or underscores. The value you specify is also applied as an optional prefix for ICL module matching and automatically take place in the tool flow when assigning related ICL modules to the new prefixed modules.

By default, the -edited_module_prefix switch puts an underscore character between the prefix and the original module name. For example, if the module name is “xyz” and the prefix is set to “abc”, the resultant module name after editing is “abc_xyz”, not “abccxyz”. When using the [set_module_matching_options -prefix_pattern_list](#) command and switch, the module prefix underscore is not assumed by default. You must explicitly specify the underscore.

Specifying a prefix is advisable when a module is used in multiple designs that are modified by Tessent Shell independently. For example, you have a module called Counter that is used in CoreA and CoreB, and those two cores are processed in isolation, adding a prefix that is a unique string to the module called Counter when editing the module in context of each core will prevent you from having different views of the same module with the same name if CoreA and CoreB are loaded together into a tool such as a simulator.

The -edited_module_prefix defaults to a null string, which means that no module name prefixing is done by default.

- **-net_uniquification_suffix *net_uniquification_suffix***

An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a net when the net that is trying to be created already exists in the design. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the net name does not already exist. For example, specifying -net_uniquification_suffix of _TS# will append the string _TS# to the net where # is replaced by the first integer above 0, making the net name unique. If net A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new net with default name A and would use B_TS2 if it tried to create a new net with default name B.

The net_uniquification_suffix defaults to "_ts#". If defined, the [create_connections -net_uniquification_suffix](#) command takes precedence.

- **-module_uniquification_suffix *module_uniquification_suffix***

An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a module. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying

-module_uniquification_suffix of _TS# will append the string _TS# to the module name where # is replaced by the first integer above 0, making the module name unique. If module A, B, and B_TS1 already exist in the design, Tesson Shell would use A_TS1 if it needed to create a new module with default name A and would use B_TS2 if it tried to create a new module with default name B.

The module_uniquification_suffix defaults to "_#".

- **-instance_uniquification_suffix *instance_uniquification_suffix***

An optional switch and string pair that defines the string that Tesson Shell uses to uniquify a leaf instance which uses the create_instance command. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying -instance_uniquification_suffix of _TS# will append the string _TS# to the instance name where # is replaced by the first integer above 0, making the leaf instance name unique. If leaf instance A, B, and B_TS1 already exist in a module, Tesson Shell would use A_TS1 if it needed to create a new instance with default name A and would use B_TS2 if it tried to create a new instance with default name B.

- **-allow_assigns *on/off***

An optional switch and literal pair that specifies that assign statements are to be converted into Verilog primitive buffers when writing out the design. For example, creating a connection between nets that are also pins causes the creation of an assign statement. Those assign statements will show up as assign statements in the output netlist unless the -allow_assigns is set to off, in which case they will be replaced by buffer primitives.

- **-inserted_object_comment_label *label* (-rtl context only)**

An optional switch and string pair that defines a label that the tool adds to comments appearing before and after each inserted object in the netlist. By default, the label is the null string "" which means no comments are inserted. For more information, refer to [Example 2](#).

- **-treat_common_ancestor_as_unique *on/off***

An optional switch and literal pair that specifies whether the create_connection command treats the common ancestor module of a one-to-many connection as unique. By default, the tool does not lift the connections as if the common ancestor is unique (that is, instantiated only once). This results in the same one-to-many connections being made in all instances of the common ancestor module. This is the setting used when instantiating the instruments as part of the process_dft_specification command.

When set to off, the connection is lifted above the common ancestor if the common ancestor is non-unique and the connection does not include all repeated instances of the destinations. This allows the remaining destinations to be connected to a different source. This functionality is only available when auto-uniquify is off.

If defined, the create_connections -treat_common_ancestor_as_unique command takes precedence.

- **-preserve_paths_for_unrolled_v2001_loops on | off**

An optional switch and literal pair that specifies whether or not to preserve SDC paths to objects found inside unrolled generate loops in Verilog 2001 design files. If you issue the [uniquify_instances](#) command on an instance that is inside a generate loop or if the [set_insertion_options -auto_uniquify_edited_modules](#) option is on, the generate loop will be unrolled as shown below based on the fact that **-preserve_paths_for_unrolled_v2001_loops** is on or off.

For a detailed discussion of the pros and cons of the two option values, refer to “[Unrolling of Generate Loops Side Effects](#)” on page 4021 in “[HDL Limitations in the Tessent Shell Flow](#)” on page 4005.

When on, each loop count is replaced by an if generate block where the block ID is the original block id with the original loop count. This block ID is escaped but the SDC path remains the same as before because in SDC paths, the escape characters are stripped out. In this format, the genvar that existed in the generate loops is no longer present so any occurrence of the genvar in the original Verilog or VHDL code is replace by the literal value of the genvar.

When you set it to off, the unrolled loops are all replaced with single count loops such that the genvar is preserved, allowing the Verilog or VHDL code that reference to the genvar to be preserved unaltered.

Original loop:

```
wire [1:0] in1, in2;
generate
genvar i;
for( i = 0; i < 2; i++ )
begin : joe
`ifndef TECHNOLOGY_32NM
    DUT io ( .A(in1[i]), .B(in2[i]), ... );
`else
    DUT1 io ( .A(in1[i]), .B(in2[i]), ... );
`endif
end for
end generate
```

Unrolled loop when **-preserve_paths_for_unrolled_v2001_loops** is on:

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
    DUT1 io ( .A(in1[0]), .b(in2[0]), ... );
end
if(1) begin : \joe[1]
    DUT1 io ( .A(in1[1]), .b(in2[1]), ... );
end
```

Unrolled loop when -preserve_paths_for_unrolled_v2001_loops is off:

```
wire [1:0] in1, in2;
genvar i;
for( i = 0; i <=0 ; i++ ) begin : \joe[0]
`ifndef TECHNOLOGY_32NM
    DUT io ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 io ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
for( i = 1; i <= 1; i++ ) begin : \joe[1]
`ifndef TECHNOLOGY_32NM
    DUT io ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 io ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
```

- **-vhdl_port_pin_net_type** bit | std_logic | std_ulogic

An optional switch and literal pair that sets the type for new ports, pins, and nets that are created in VHDL. This option has no default. If the option is specified, then new ports, pins, and nets will get the specified type with two exceptions:

- If a new signal is connected to an output pin, then the type of the signal will be the same as the type of the pin.
- If a new signal is connected to another signal with a user-defined type, then the type of the new signal will be the same as the type of the other signal.

If the option is not specified, then the types of new ports, pins, and nets will be determined as follows:

- If the port, pin, or net is to be connected to an existing VHDL signal, then the type of the new signal will be governed by the type of the existing signal to which it is to be connected.
- If the port, pin, or net will not be connected to an existing VHDL signal, then the type of the new signal will be 'bit'.

- **-open_input_pins** tie0 | filler_nets

An optional switch and literal pair that specifies how Tesson Shell handles unconnected input pins for inserted objects. When the option tie0 is specified, the unconnected inputs will be tied low. When filler_nets is specified, the unconnected inputs are tied to a common filler net that is not driven.

- **-remove_inactive_sections_in_unrolled_generate_loops** {on | off}

An optional switch and literal pair that removes inactive parts of loops if the loop contains a part that is inactive for certain values of the loop index.

The switch has only an effect when a VHDL loop is unrolled. Use this option to avoid parsing errors with the inactive section that can happen when the index value is substituted for the unrolled loop.

- **-trace_across_module_boundaries_to_find_functional_drivers {on | off}**

An optional switch and literal pair that specifies whether the tool traces across module boundaries to identify the driver for the [has_functional_source](#) pin attribute. The default is off.

When set to on, the tool continues tracing across module boundaries until it finds a cell driver, an RTL driver, or an open or a tie value to determine the value of the attribute.

Examples

Example 1

The following example sets various insertion options that will affect the behavior of the insertion commands.

```
SETUP> set_insertion_options -edited_module_prefix MyCoreName_ \
           -module_unification_suffix _uniq# -allow_assigns Off
```

Example 2

The following example sets the comment label for inserted objects to “myFlow”.

```
SETUP> set_insertion_options -inserted_object_comment_label myFlow
```

A resulting netlist would then look like this:

```
module top(a, z, b);
    input a;
    output z;
    /* myFlow Ports Start */
    inout b;
    wire b;
    /* myFlow Ports End */
```

Example 3

The following example deactivates automatic commenting of inserted objects. This is the same as the default behavior if you do not specify the `-inserted_object_comment_label` argument at all.

```
SETUP> set_insertion_options -inserted_object_comment_label {}
```

Related Topics

[get_insertion_option](#)

set_insert_test_logic_options

Context: dft -scan, dft -test_points

Mode: setup, analysis

Sets new prefix/infix values that will be used when logic is created by the insert_test_logic command.

Usage

```
set_insert_test_logic_options
  [ -inserted_object_prefix inserted_object_prefix ]
  [ -persistent_cell_prefix persistent_cell_prefix ]
  [ -persistent_clock_cell_prefix persistent_clock_cell_prefix ]
  [ -logic_type
    [ generic
      | clock_gater_fix
      | clock_control_fix
      | dedicated_input_wrapper
      | dedicated_input_holding_wrapper
      | dedicated_output_wrapper
      | lockup
      | pipeline
      | ram_fix
      | testpoint_control
      | testpoint_observation
      | tri_state_fix
      | x_bounding ] ]
  -logic_infix logic_infix ]
```

Description

The set_insert_test_logic_options command sets new prefix/infix values that will be used when logic is created by the insert_test_logic command.

Arguments

- **-inserted_object_prefix *inserted_object_prefix***
An optional switch and string pair that specifies the prefix of HDL objects created during “insert_test_logic”. The default is “ts_”.
- **-persistent_cell_prefix *persistent_cell_prefix***
An optional switch and string pair that specifies the prefix of persistent cell instances created during “insert_test_logic”. The default is “tessent_persistent_cell_”.
- **-persistent_clock_cell_prefix *persistent_clock_cell_prefix***
An optional switch and string pair that specifies the prefix of persistent clock cell instances created during “insert_test_logic”. The default is “tessent_persistent_cell_”.

- **-logic_type**

An optional switch and literal pair that specifies the logic type for which the infix must be set. The possible logic types are as follows:

generic — Used for generic logic and has the default infix “logic_”.

clock_gater_fix — Used for clock gater control and has the default infix “cgfix_”.

clock_control_fix — Used for clock control and has the default infix “clkfix_”.

dedicated_input_wrapper — Used for dedicated input wrapper cells and has the default infix “diw_”.

dedicated_input_holding_wrapper — Used for dedicated input holding wrapper cells and has the default infix “dihw_”.

dedicated_output_wrapper — Used for dedicated output wrapper cells and has the default infix “dow_”.

lockup — Used for scan chain retiming cells and has the default infix “lockup_”.

pipeline — Used for scan chain pipeline cells has the default infix “pipeline_”.

ram_fix — Used for ram control and has the default infix “ramfix_”.

testpoint_control — Used for control testpoint and has the default infix “cp_”.

testpoint_observation — Used for observation testpoint and has the default infix “op_”.

tri_state_fix — Used for tristate and bidi control and has the default infix “trifix_”.

x_bounding — Used for X-bounding and has the default infix “xb_”.

Note

 The -logic_type switch must be used in conjunction with the -logic_infix switch.

- **-logic_infix *logic_infix***

An optional switch and string pair that specifies the infix to use when implementing HDL objects of a specified logic type during “insert_test_logic”.

Note

 The -logic_infix switch must be used in conjunction with the -logic_type switch.

Examples

The following example demonstrates the use of the set_insert_test_logic_options command.

```
SETUP> get_insert_test_logic_option -logic_type dedicated_input_wrapper
diw_
SETUP> set_insert_test_logic_options -logic_type dedicated_input_wrapper -logic_infix
inreg_
SETUP> get_insert_test_logic_option -logic_type dedicated_input_wrapper
inreg_
```

SETUP> report_insert_test_logic_options

// 'insert_test_logic' Command Option	Current Value
// -----	-----
// -inserted_object_prefix	ts_
// -logic_type clock_control_fix	clkfix_
// -logic_type clock_gater_fix	cfgfix_
// -logic_type dedicated_input_holding_wrapper	dihw_
// -logic_type dedicated_input_wrapper	inreq_
// -logic_type dedicated_output_wrapper	dow_
// -logic_type generic	logic_
// -logic_type lockup	lockup_
// -logic_type pipeline	pipeline_
// -logic_type ram_fix	ramfix_
// -logic_type testpoint_control	cp_
// -logic_type testpoint_observation	op_
// -logic_type tri_state_fix	trifix_
// -logic_type x_bounding	xb_
// -persistent_cell_prefix	tessent_persistent_cell_
// -persistent_clock_cell_prefix	tessent_persistent_clock_cell_

Related Topics

[get_insert_test_logic_option](#)
[report_insert_test_logic_options](#)

set_internal_fault

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies whether the tool allows faults within or only on the boundary of library models.

Usage

`set_internal_fault OFF | ON`

Description

Specifies whether the tool allows faults within or only on the boundary of library models.

The `set_internal_fault` command specifies whether the tool allows faults on internal nets of library models (internal faulting) or only on the library model boundary. The default upon invocation of the tool is to allow faults only on the library model boundary.

Note

 This command is not valid for the bridge logic fault model (specified with the “[“set_fault_type Bridge -Static” command](#)”).

Arguments

- **OFF**
A required literal that allows faults only on the boundary of the library models. This is the default.
- **ON**
A required literal that allows faults on the internal nets of library models.

Examples

The following example performs ATPG using a fault list that includes internal nets within the library cells. These internal nets will be shown as fault sites when the `report_faults` command is executed.

```
set_context patterns -scan
set_internal_fault on
set_system_mode analysis
create_patterns
report_faults -all
```

set_internal_name

Context: all contexts

Mode: setup

Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.

Usage

`set_internal_name OFF | ON`

Description

Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.

The `set_internal_name` command specifies whether to keep internal library pins with no-fault attributes. Normally, you should delete these names for memory and performance reasons. The default operation (OFF) upon invocation is to delete these names.

Arguments

- **OFF**
A literal that deletes the lowest level pin names if they have the `nofault` attribute. This is the default.
- **ON**
A literal that keeps the lowest level pin names, even if they have the `nofault` attribute.

Examples

The following example deletes pin names with the `nofault` attribute.

`set_internal_name off`

set_io_mask

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Modifies the behavior of bidirectional pins so that their expected values will always be X during test cycles in which the primary input portion of the bidirectional pin is being forced.

Usage

`set_io_mask OFF | ON`

Description

Modifies the behavior of bidirectional pins so that their expected values will always be X during test cycles in which the primary input portion of the bidirectional pin is being forced.

Typically, when the tool forces stimulus on an bidirectional pin, it will expect to measure the same value on the corresponding primary output pin. This command lets you modify this behavior in cases where this is undesirable.

Arguments

- OFF
An optional literal that disables the ability to mask the bidirectional pin output value. This is the default.
- ON
An optional literal that enables the ability to mask the bidirectional pin output value.

Examples

The following example causes the tool to mask the bidirectional pin output value.

`set_io_mask on`

[set_latch_handling](#)

Context: dft -scan

Mode: setup

Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.

Usage

`set_latch_handling None | Scan`

Description

Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.

The `set_latch_handling` command specifies whether the tool can consider non-transparent latches as candidates for scan insertion. If you determine that the tool is to consider non-transparent latches as candidates for scan insertion, you must turn on the appropriate [`set_test_logic`](#) command settings before the tool performs the rules checking process.

If you use the [`set_drc_handling`](#) command to set the D6 rule to error or warning, D6 checks the non-scannable latches for transparency. By default, if they are not transparent and you turned test logic insertion on, the tool does not consider the non-transparent latches for scan insertion. However, if you use the Scan argument with the [`set_latch_handling`](#) command, the tool will consider the non-transparent latches for scan insertion.

If you use the [`set_drc_handling`](#) command to ignore the D6 rule, then rules checking does not check the non-scannable latches for transparency and the tool will automatically consider all non-scannable latches, whose test logic you turned on, for scan insertion.

Arguments

- **None**

A literal that specifies to give no consideration to non-transparent latches for scan insertion. This is the default upon invocation of the tool.

- **Scan**

A literal that specifies to consider non-transparent latches for scan insertion when test logic is turned on.

Related Topics

[`set_drc_handling`](#)

[`set_test_logic`](#)

set_layout_core_instance

Context: patterns -scan_diagnosis

Mode: setup, analysis

Sets the current layout core instance for inter-scan cell logic reporting with the report_scan_polygons command.

Usage

`set_layout_core_instance instance_path`

Description

Sets the current layout core instance for inter-scan cell logic reporting with the report_scan_polygons command.

You must use this command when you perform inter-scan cell logic reporting in the context of hierarchical layout-aware diagnosis. The command is required because inter-scan cell logic reporting occurs outside the processing of a diagnosis report. The set_layout_core_instance command has no effect on diagnose_failures because the tool automatically sets the layout core instance based on the coreInstance keyword in the tracking_info section of the failure file when processing a diagnosis report.

See “[Diagnosis for Hierarchical Design](#)” and “[Inter-Scan Cell Polygon Reporting for Chain Diagnosis](#)” in the *Tessent Diagnosis User’s Guide* for more information.

Arguments

- ***instance_path***

A required string that specifies the path to a layout core instance.

Related Topics

[get_layout_core_instance](#)

[report_scan_polygons](#)

set_lbist_controller_options

Context: dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies global options to configure the LogicBIST controller.

Usage

IP Creation Usage

```
set_lbist_controller_options [off | on] -max_shift_cycles {integer1 [integer2]}
  -max_capture_cycles {integer1 [integer2]}
  -max_pattern_count {integer1 [integer2]}
  [-max_warmup_pattern_count {integer1 [integer2]})
  [-controller_chain_mode {off | on} -controller_chain_clock {tck | edt_clock}]
  -capture_procedures {procname [integer%] | count | integer%}...
  [-shift_counter_resolution {bit | byte}]
  [-pre_post_shift_dead_cycles integer]
  [-tessent_occ {auto | on}] [-burn_in {off | on}]
  [-single_chain_mode_skip_edt_blocks {off | on}]
```

Fault Simulation Usage

```
set_lbist_controller_options [off | on] -capture_procedures {procname integer%}...
```

Description

Specifies global options to configure the LogicBIST controller.

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- off | on

An optional literal that specifies enabling or disabling LogicBIST controller configuration. The default is on.

- **-max_shift_cycles** {*integer1* [*integer2*]}

A required switch and integer pair or triplet that specifies the maximum possible shift length the tool uses to synthesize the LogicBIST shift length counter.

An optional second argument, *integer2*, can be used to specify the value to be loaded in the shift length counter for the hardware default mode of operation. This value should be less than or equal to the *integer1*, the maximum shift cycles.

If you do not specify a value, then the maximum shift cycles value will be used for the hardware default mode.

- **-max_capture_cycles {integer1 [integer2]}**

A required switch and integer pair or triplet that specifies the maximum possible capture width the tool uses to synthesize the LogicBIST capture width counter.

An optional second argument, *integer2*, can be used to specify the value to be loaded in the capture width counter for the hardware default mode of operation. This value should be less than or equal to the **integer1**, the maximum capture width.

If you do not specify a value, then the maximum capture width will be used for the hardware default mode.

- **-max_pattern_count {integer1 [integer2]}**

A required switch and integer pair or triplet that specifies the maximum number of LogicBIST patterns that will be applied when the tool synthesizes the LogicBIST pattern counter.

An optional second argument, *integer2*, can be used to specify the value to be loaded in the pattern counter for the hardware default mode of operation. This value should be less than or equal to the **integer1**, the maximum pattern count.

If you do not specify a value, then the maximum pattern count will be used for the hardware default mode.

- **-max_warmup_pattern_count {integer1 [integer2]}**

An optional switch and integer pair that specifies the maximum number of Logic BIST warm-up patterns that the tool can apply. The default is 511. *integer1* gets rounded up to the next power of 2 minus 1.

You can use an optional second argument, *integer2*, to specify the number of warm-up patterns to be applied for the hardware default mode of operation. This value must be less than or equal to *integer1*, the maximum pattern count. If you do not specify a value, then no warm-up patterns will be applied for the hardware default mode.

If you do not want the tool to insert the warm-up pattern count register in the logic BIST controller, set *integer1* to 0. Refer to “[Warm-Up Patterns](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for details.

- **-controller_chain_mode {off|on}**

An optional switch and literal pair that specifies whether to enable or disable the Controller Chain Mode. When enabled, the tool creates an RTL scan chain so that it can calculate test coverage for the LBIST controller and EDT/LBIST blocks. The default is off.

- **-controller_chain_clock {tck | edt_clock}**

An optional switch and literal pair that specifies whether the tool will use the tck or edt_clock clock during Controller Chain Mode. The default is edt_clock.

- **-capture_procedures {procname [integer%] | count | integer%}**

A required switch and repeatable string pair that specifies the Named Capture Procedure (NCP) name for which the NCP should be applied.

If a percentage is specified, all the NCPs are repeated according to the specified percentage once every 256 patterns. The sum of all the specified percentages should add up to 100.

You can specify the percentage in addition to the name of the NCP or the percentage on its own without naming the NCP.

Because the integer percentage may not be known at IP creation time, the integer percentage can be omitted and specified during fault simulation. At that time, you can program the percentage activity for each NCP without having to regenerate the IP. A percentage value of 0 means that the NCP will not be active for the pattern set.

The mutually-exclusive **count** option generates the LBIST controller that can support up to the "count" NCPs.

- **-shift_counter_resolution {bit | byte}**

An optional switch and literal pair that specifies whether the shift counter allows shift lengths that operate in multiples of 8 or 1. When you specify "byte", the tool rounds up the length of the longest scan chain to the next multiple of 8. For example, if the longest scan chain is 67 flops long, then the tool applies 72 shifts. This is the default.

By default, the tool applies the number of shifts equal to the length of the longest scan chain.

Note

If the shift length specified during IP generation differs from that used during fault simulation, you may be required to use the **set_number_shifts** command during fault simulation when creating HW default patterns. For example, suppose you have 29 scan chains for fault simulation, but for IP generation you specified a HW default max shift of 38. In this case you would specify the **set_number_shifts** command to create patterns for HW default mode.

- **-pre_post_shift_dead_cycles integer**

An optional switch and integer pair that specifies how many shift pause and capture pause dead cycles, at the shift clock period, will be applied before and after the capture clocks are pulsed. The specified integer can be between 2 and 8, with the default being 8. For example, specifying 5 means that the shift pause will consist of 5 dead cycles and the capture pause will consist of 5 dead cycles.

Reducing the number of pause cycles reduces the test time, but it may also make it harder to achieve timing closure of the test logic. To ensure that the shift and capture signals are staggered an integer value of 1 is not supported.

- **-tesson_occ {auto | on | off}**

An optional switch and literal that generates a Tessent OCC-compatible LBIST controller for skeleton designs that do not contain Tessent OCC instances. You specify this switch during pre-synthesis IP creation. Choose one of the following options:

auto — Infers the Tessent OCC instruments from the TCD and the ICL. This is the default.

on — Enables generation of Tessent OCC compatible LBIST controller when OCC core instances are not added.

- **-burn_in {off | on}**

An optional switch and literal pair that specifies whether to enable or disable support for wafer-level burn-in tests. When enabled, the tool creates additional RTL to allow LogicBIST to run for longer periods of time than supported by the pattern counter hardware. The default is off.

- **-single_chain_mode_skip_edt_blocks {off | on}**

An optional switch and literal pair that specifies whether to insert a SIB in each EDT block that allows optionally skipping all scan chains in a given EDT block during single chain unload. The default is on. For details, refer to “[Single Chain Mode Diagnosis](#)” in the *Hybrid TK/LBIST Flow User’s Manual*.

Examples

Example 1

The following example configures the LogicBIST controller for a maximum scan chain length of 1000 and a maximum capture width of 8 cycles. The maximum number of patterns is set to 250,000 and number of patterns run in the hardware default mode is set as 5,000. There are 4 NCPs defined for LogicBIST test such that each of them is active for 25% of the patterns. A maximum of 127 warm-up patterns will be allowed with 16 warm-up patterns run in the hardware default mode.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8
set_lbist_controller_options -max_pattern_count {250000 5000}
set_lbist_controller_options -max_warmup_pattern_count {128 16}
set_lbist_controller_options -capture_procedures \
    {clkseq1 25 clkseq2 25 clkseq3 25 clkseq4 25}
```

Example 2

The following example is the same as the previous example except that the NCP percentage integer values have not been specified with the commands issued for IP creation. In this case, the NCP percentage integer values must be set by issuing the command with the desired values for fault simulation.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8
set_lbist_controller_options -max_pattern_count {250000 5000}
set_lbist_controller_options -capture_procedures {clkseq1 clkseq2 clkseq3 clkseq4}
```

Related Topics

[report_lbist_configuration](#)
[set_clock_controller_pins](#)
[set_edt_options](#)
[set_lbist_instances](#)

set_lbist_instances

Context: dft -logic_bist

Mode: setup

Specifies the instance in which the LogicBIST controller or single chain mode logic is placed.

Usage

```
set_lbist_instances [-controller_location instance_path_name]  
                  [-single_chain_mode_logic_location instance_path_name]
```

Description

Specifies the instance in which the LogicBIST controller or single chain mode logic is placed.

This command is used with the hybrid TK/LBIST flow. Refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

By default, the tool places the LogicBIST controller and single chain mode logic in the top level of the design. Using this command, you can specify a different location for the LogicBIST controller or single chain mode logic.

For example, you can use this command to place the LogicBIST controller or single chain mode logic instance in a specific test logic block or in an always-ON block on a design with multiple power domains.

Arguments

- **-controller_location *instance_path_name***
An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the LogicBIST IP.
- **-single_chain_mode_logic_location *instance_path_name***
An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the single chain mode logic.

Examples

The following example instantiates the LogicBIST controller in an always ON power domain instance */pwr_always/dft_logic*:

```
set_lbist_instances -controller_location /pwr_always/dft_logic
```

The following example instantiates the LogicBIST controller and single chain mode logic in a dedicated test logic block instance */dft_test_logic_i*:

```
set_lbist_instances -controller_location /dft_test_logic_i \  
                   -single_chain_mode_logic_location /dft_test_logic_i
```

Related Topics

[set_edt_options](#)

[set_clock_controller_pins](#)

set_lbist_pins

Context: dft -logic_bist

Mode: setup

Specifies the connection information for LogicBIST controller pins and enables renaming LogicBIST pins.

Usage

Specify LogicBIST Controller Pin Connections

```
set_lbist_pins
    scan_en {{top_pin_name | -} [internal_pin_name | -]}... |
    clock {top_pin_name | -} [internal_pin_name] |
    tck {top_pin_name | -} [internal_pin_name] |
    setup_shift_scan_in {top_pin_name | -} [internal_pin_name] |
    setup_shift_scan_out {top_pin_name | -} [internal_pin_name] |
    tap_instruction_decode {top_pin_name | -} [internal_pin_name] |
    test_logic_reset {top_pin_name | - [internal_pin_name]} [-active high | low] |
    capture_dr {top_pin_name | -} [internal_pin_name] |
    shift_dr {top_pin_name | -} [internal_pin_name] |
    update_dr {top_pin_name | -} [internal_pin_name] |
    xbounding_en {pin_name... | -no_connection} |
    control_point_en {pin_name... | -no_connection} |
    observe_point_en {pin_name... | -no_connection} |
    test_en {pin_name... | -no_connection} |
    mcp_bounding_en {pin_name... | -no_connection} |
    controller_chain_en {{top_pin_name | -} [internal_pin_name | -]} |
    controller_chain_scan_in {{top_pin_name | -} [internal_pin_name | -]} |
    controller_chain_scan_out {{top_pin_name | -} [internal_pin_name | -]} |
    wrapper_chain_input_scan_en {{top_pin_name | -} [internal_pin_name | -]}...
```

Rename LogicBIST Pins

```
set_lbist_pins [pin_function port_name -on_controller_module] ...
```

Description

Specifies the connection information for LogicBIST controller pins. You can also use this command to rename pins at the LogicBIST module level in the RTL.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Specifying LogicBIST Pin Connections

When specifying LogicBIST pin connections, the pins are connected to the single top-level LogicBIST controller. This is in a global context and not specific to the current EDT block. The

first argument is a literal string that indicates the pin being described in the command followed by one or more pin names.

If you issue this command multiple times with the same pin type, then the last pin or pins you specify are used. To specify multiple pin names corresponding to a pin type, specify all the names in a single issuance of the command.

If the signal to be connected to the LogicBIST controller is internally generated and does not have a corresponding top-level pin, use a '-' literal in place of the *top_pin_name* argument.

Renaming Pins

When renaming pins, the *-on_controller_module* switch accepts the following pin functions. The default pin names are shown. By default, the done pin is not created on the LogicBIST controller. Only when this pin is specified will the port be created on the RTL.

Note

 Pin functions shown with asterisks (*) are those for which you can also specify the pin connections.

Table 6-12. Pin Functions That Can Be Renamed

Pin Function	Default Name	Pin Function	Default Name
clock *	shift_clock_src	controller_chain_scan_out *	ccm_scan_out
tck *	ijtag_tck	controller_chain_scan_en	ccm_scan_en
setup_shift_scan_in *	ijtag_si	test_clock	test_clock
setup_shift_scan_out *	ijtag_so	edt_update	edt_update
tap_instruction_decode *	ijtag_sel	clock_out	edt_lbist_clock
test_logic_reset *	ijtag_reset	reset_out	lbist_reset
capture_dr *	ijtag_ce	enable_out	lbist_en
shift_dr *	ijtag_se	done	<i>no default value</i>
update_dr *	ijtag_ue	prpg_en	lbist_prpg_en
scan_en *	scan_en_in scan_en_out	mistr_en	mistr_accumulate_en
controller_chain_en *	ccm_en	ijtag_select_out	edt_sib_en

Table 6-12. Pin Functions That Can Be Renamed (cont.)

Pin Function	Default Name	Pin Function	Default Name
controller_chain_scan_in *	ccm_scan_in		

Arguments**Specify LogicBIST Controller Pin Connections**

- ***top_pin_name* | -**

A required string or literal that specifies the pin name to be connected to the LogicBIST controller. Choose from the following:

 - top_pin_name* — A string that specifies the name of the pin to assign to the specific signal.
 - — A literal used for internally-generated pins and do not have a corresponding top-level pin.
- ***internal_pin_name***

An optional string that specifies an internal pin name.
- **scan_en {*top_pin_name* | -} [*internal_pin_name* | -]}**

A required literal and string pair that specifies the scan enable pin and optionally the internal pin corresponding to the pad output. Multiple values can be specified if the design has multiple scan enables.
- **clock {*top_pin_name* | -} [*internal_pin_name*]**

An optional literal and string pair that specifies the LogicBIST clock pin.

When a top-level tester controllable clock is available, the same pin can be used for both EDT and LogicBIST clocks. However, when the EDT clock is internally generated, a different asynchronous free-running clock should be specified as the LogicBIST clock.
- **tck{*top_pin_name* | -} [*internal_pin_name*]**

An optional literal and string pair that specifies the JTAG TCK input and optionally the internal pin corresponding to the pad output.
- **setup_shift_scan_in {*top_pin_name* | -} [*internal_pin_name*]**

An optional literal and string pair that specifies the connection to the seeding register input. In a JTAG based design, this would typically be connected to the TDI pin.
- **setup_shift_scan_out {*top_pin_name* | -} [*internal_pin_name*]**

An optional literal and string pair that specifies the connection to the seeding register output. In a JTAG based design, this would typically be connected to the TDO pin through the DR-mux.

- **tap_instruction_decode** {*top_pin_name* |-} [*internal_pin_name*]

An optional literal and string pair that specifies the connection to the instruction decoder output that indicates when the TAP instruction that targets LogicBIST is loaded in the JTAG instruction register.

- **test_logic_reset** {*top_pin_name* |-} [*internal_pin_name*] [-active high | low]

An optional literal and string pair that specifies the connection to the test_logic_reset output of the TAP controller that indicates when the TAP is in test-logic-reset state. The default active reset state is low. Use the optional -active switch to change the active reset state to high.

- **capture_dr** {*top_pin_name* |-} [*internal_pin_name*]

An optional literal and string pair that specifies the connection to the capture_dr output of the TAP controller that indicates when the TAP is in the capture-DR state.

- **shift_dr** {*top_pin_name* |-} [*internal_pin_name*]

An optional literal and string pair that specifies the connection to the shift_dr output of the TAP controller that indicates when the TAP is in the shift-DR state.

- **update_dr** {*top_pin_name* |-} [*internal_pin_name*]

An optional literal and string pair that specifies the connection to the update_dr output of the TAP controller that indicates when the TAP is in the update-DR state.

- **xbounding_en** {*pin_name* | -no_connection}

An optional literal and string pair that specifies the xbounding enable pins. Specifying the -no_connection option instead of *pin_name* creates the pin in the LogicBIST controller but does not connect it to the design.

- **control_point_en** {*pin_name* | -no_connection}

An optional literal and string pair that specifies the control point enable pins. The pin names refer to the control_point_en pin created by Tessent Shell during test point insertion.

Specifying the -no_connection option instead of *pin_name* creates the pin in the LogicBIST controller but does not connect it to the design.

- **observe_point_en** {*pin_name* | -no_connection}

An optional literal and pin name that specifies the observe point enable pins. The pin names refer to the observe_point_en pin created by Tessent Shell during test point insertion.

Specifying the -no_connection option instead of *pin_name* creates the pin in the LogicBIST controller but does not connect it to the design.

- **test_en** {*pin_name* | -no_connection}

An optional literal and string pair that specifies the test enable pins. The pin names refer to the test_en pin created during scan insertion. Specifying the -no_connection option instead of *pin_name* creates the pin in the LogicBIST controller but does not connect it to the design.

- **mcp_bounding_en** {*pin_name* | -no_connection}
An optional literal and string pair that specifies the MCP bounding enable pins. The pin names refer to the *mcp_bounding_en* pin created during X-bounding to bound false and multicycle paths. Specifying the -no_connection option instead of *pin_name* creates the pin in the LogicBIST controller but does not connect it to the design.
- **controller_chain_en** {{*top_pin_name* | -} [*internal_pin_name* | -]}
An optional literal and string pair that specifies the controller chain mode enable pin and optionally the internal pin corresponding to the pad output.
- **controller_chain_scan_in** {{*top_pin_name* | -} [*internal_pin_name* | -]}
An optional literal and string pair that specifies the controller chain mode scan input pin and optionally the internal pin corresponding to the pad output.
- **controller_chain_scan_out** {{*top_pin_name* | -} [*internal_pin_name* | -]}
An optional literal and string pair that specifies the controller chain mode scan output pin and optionally the internal pin corresponding to the pad output.
- **wrapper_chain_input_scan_en** {{*top_pin_name* | -} [*internal_pin_name* | -]}
An optional literal and string pair that specifies the scan enable pin of the input wrapper chains and optionally the internal pin corresponding to the pad output. Multiple values can be specified if the design has multiple input scan enables for the wrapper chain. The wrapper chain input scan enable signals are held at constant 1 (specifically, shift) during LogicBIST test. For ATPG, they remain controllable from specified top or internal pins.

Rename LogicBIST Pins

- ***pin_function***
A required string for pin renaming that specifies the pin function whose default name you want to change. Refer to [Table 6-12](#).
- ***port_name***
A required string for pin renaming that specifies the new name for the specified pin function.
- **-on_controller_module**
A required switch for pin renaming that specifies to rename the specified pin on the LogicBIST module in the LogicBIST RTL. Pin renaming only applies to the specified pin at the LogicBIST module level.

Examples

Example 1

This example specifies the connections between design pins and LogicBIST controller pins, including the TAP interface. A dash is used as the first argument for the TAP generated control signals, since there is no corresponding top-level pin for these signals.

```
set_ibist_pins clock refclk
```

```
set_lbist_pins scan_en scan_en
set_lbist_pins xbounding_en lbist_en
set_lbist_pins setup_shift_scan_in tdi
set_lbist_pins setup_shift_scan_out { tdo jtag/scanCfgReg_so }
set_lbist_pins shift_dr { - jtag/shift_dr }
set_lbist_pins capture_dr { - jtag/capture_dr }
set_lbist_pins update_dr { - jtag/update_dr }
set_lbist_pins test_logic_reset { - jtag/tlr }
set_lbist_pins tap_instruction_decode { - jtag/scanCfgReg_en }
```

Example 2

In this example, the design pin “lbist_en” is used as the enable signal for x-bounding, control points, and observe points. Note that the same pin is specified for all three pin types. The LogicBIST controller will have only a single enable register named “x_bounding_en.”

```
set_lbist_pins xbounding_en lbist_en
set_lbist_pins control_point_en lbist_en
set_lbist_pins observe_point_en lbist_en
```

Example 3

This example illustrates how to specify multiple scan enable signals without pads. Since the scan enable pin names should be specified in pairs, a dash is used for each of the non-existent internal pin names.

```
set_lbist_pins scan_en {scan_en - wrp_scan_en -}
```

Example 4

The following example generates TDR bits inside the LogicBIST controller for all different enables signals for a skeleton flow in which the design enable pins are not yet available:

```
set_lbist_pins xbounding_en -no_connection
set_lbist_pins control_point_en -no_connection
set_lbist_pins observe_point_en -no_connection
set_lbist_pins test_en -no_connection
set_lbist_pins mcp_bounding_en -no_connection
```

Example 5

The following example renames LogicBIST ports back to their pre-2018.1 release versions for purposes of backwards annotation.

```
set_lbist_pins tck tck -on_controller_module
set_lbist_pins setup_shift_scan_in lbist_scan_in -on_controller_module
```

```
set_lbist_pins setup_shift_scan_out lbist_scan_out -on_controller_module
set_lbist_pins tap_instruction_decode sib_en -on_controller_module
set_lbist_pins test_logic_reset sib_reset -on_controller_module
set_lbist_pins capture_dr sib_capture_en -on_controller_module
set_lbist_pins shift_dr sib_shift_en -on_controller_module
set_lbist_pins update_dr sib_update_en -on_controller_module
set_lbist_pins edt_clock edt_clock -on_controller_module
set_lbist_pins test_clock edt_clock -on_controller_module
set_lbist_pins clock_out lbist_clock -on_controller_module
set_lbist_pins prpg_en prpg_en -on_controller_module
set_lbist_pins misr_en misr_en -on_controller_module
set_lbist_pins occ_shift_clock clk_ctrl_shift_clk -on_controller_module
set_lbist_pins occ_scan_en clk_ctrl_scan_en -on_controller_module
set_lbist_pins occ_capture_en clk_ctrl_capture_en -on_controller_module
```

Related Topics

[set_lbist_controller_options](#)
[report_lbist_pins](#)
[set_edt_options](#)

set_lbist_power_controller_options

Context: dft -logic_bist

Mode: setup

Specifies creation of the low-power shift controller for LogicBIST.

Usage

```
set_lbist_power_controller_options [shift {none | disabled | enabled}]  
[-min_switching_threshold_percentage sw_percentage]]
```

Description

Specifies creation of the low-power shift controller for LogicBIST.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

This command is applicable to the current LogicBIST block.

When you specify this command during IP creation, the tool creates the low-power shift controller for LogicBIST. This command is carried forward in the generated fault simulation dofile to indicate whether the current EDT block has low-power shift logic synthesized (automatically added to the fault simulation dofile).

Arguments

- **shift none | disabled | enabled**

A literal pair that indicates the configuration parameters for controlling shift power.

Choose one of the following:

- **none** — Indicates that the low-power controller is not synthesized.
- **disabled** — Forces a LogicBIST run to bypass the low power controller.
- **enabled** — Enables the low-power controller.

When you specify either 'enabled' or 'disabled', a low-power controller with the specified switching threshold percentage is synthesized.

- **-min_switching_threshold_percentage *sw_percentage***

An optional switch and integer that specifies the switching threshold value for the hardware default mode. Valid integer values are between 1 and 50. The default value is 15 percent.

During fault simulation, you can specify any switching threshold.

Examples

This example shows three EDT/LogicBIST blocks, two of which are to be synthesized with both EDT and LogicBIST low-power logic, while the third block does not have low power shift logic.

```
add_edt_block lpblk1
set_edt_power_controller shift enabled -min_switching_threshold 20
set_ibist_power_controller_options shift enabled
add_edt_block lpblk2
set_edt_power_controller shift enabled -min_switching_threshold 8
set_ibist_power_controller_options shift enabled
add_edt_block rgblk
```

Related Topics

[set_power_control](#)
[set_edt_power_controller](#)

set_learn_report

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Specifies whether the report_gates command can display the learned behavior for a specific gate.

Usage

set_learn_report OFF | ON

Description

Specifies whether the report_gates command can display the learned behavior for a specific gate.

The set_learn_report command specifies whether the report_gates command should include the information that the tool collects during the static learning process. The application automatically performs the static learning process immediately after it flattens the simulation model, which happens when you leave the Setup mode or issue the [create_flat_model](#) command. The static learning process provides general information on the design that the tool can then use in speeding up the ATPG process (such as values that are impossible on other gates if the selected gate is at a specific value.)

Once you enable access to the static learned information with the set_learn_report command, you can specify for the tool to display the learned information on a selected gate by using the report_gates command.

While you can also access the learned information with the report_gates command by using the -Type option, this method displays the information for all the gates of the specified gate type. When you enable access with the set_learn_report command, the tool automatically displays the learned information with all command options. Therefore, you can restrict the report to the learned information on an individual object.

Arguments

- OFF
An optional literal that disables access to the learned behavior information. This is the default.
- ON
An optional literal that enables access to the learned behavior information.

Examples

The following example enables access to the learned behavior and then accesses that information:

```
set_learn_report on
report_gates 28

/MX3/OR1 (28) OR
    IO      I  20-/MX3/AN2/OUT
    I1      I  24-/MX3/AN1/OUT
    OUT     O  37-/OUT0
Learned behavior:  MUX(9,13,17)
```

Related Topics

[report_gates](#)

set_lfsr_seed

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Sets the initial value of a PRPG or a MISR.

Usage

`set_lfsr_seed name seed [-Binary | -Hexadecimal]`

Description

Sets the initial value of a PRPG or a MISR.

The `set_lfsr_seed` command initializes the value of a PRPG or MISR with a seed value. The seed value must be specified by a hexadecimal or binary number greater than zero.

Arguments

- ***name***
A required string that specifies the name of the PRPG or MISR.
- ***seed***
A required number, greater than zero, that specifies the initial state of the LFSR.
- **-Binary**
An optional switch that specifies to output the value of the *seed* argument in binary format. By default, the seed is output in hexadecimal.
- **-Hexadecimal**
An optional switch that specifies to output the value of the *seed* argument in hexadecimal format. This is the default.

Examples

`set_lfsr_seed prpg 5`

Related Topics

[add_lfsr_taps](#)

[add_lfsrs](#)

[delete_lfsrs](#)

set_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Changes the *shift_type* and *tap_type* default setting for the add_lfsrs and add_lfsr_taps commands.

Usage

```
set_lfsrs [-Both | -Serial | -Parallel] [-Out | -In | -Ca]
```

Description

Changes the *shift_type* and *tap_type* default setting for the add_lfsrs and add_lfsr_taps commands.

The set_lfsrs command controls the default setting for the *shift_type* and *tap_type* switches. You specify the LFSR's shift technique by using one of the following *shift_type* switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following *tap_type* switches: -Out or -In; or you specify the type of LFSR as cellular automata using the *tap_type* switch -Ca. When you change one or both of the default settings, all future add_lfsrs and add_lfsr_taps commands use the new default.

Arguments

The following lists the three *shift_type* switches of which you can choose only one:

- **-Both**

An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default behavior upon invocation of the tool.

- **-Serial**

An optional switch specifying that a serial shift LFSR shifts a number of times equal to the length of the longest scan chain for each scan pattern.

- **-Parallel**

An optional switch specifying that a parallel shift LFSR shifts once for each scan pattern.

The following lists the three *tap_type* switches of which you can only choose one:

- **-Out**

An optional switch that places the exclusive-or taps outside the register path. This is the default upon invocation of the tool.

- **-In**

An optional switch specifying that the exclusive-or taps are placed outside the register path.

- **-Ca**

An optional switch specifying that the default PRPG type is cellular automata.

Examples

The following example changes the default *shift_type* setting to Serial and the default *tap_type* switch to In:

```
set_lfsrs -serial -in
add_lfsrs lfsr1 prpg 5 13
add_lfsrs lfsr2 prpg 5 11
add_lfsr_taps lfsr1 2 3
add_lfsr_taps lfsr2 3 4
set_system_mode analysis
```

Related Topics

[add_lfsrs](#)
[add_lfsr_taps](#)
[delete_lfsrs](#)
[delete_lfsr_taps](#)
[report_lfsrs](#)

set_license_queue_timeout

Context: all contexts

Mode: all modes

Specifies the amount of time that the tool queues for one or more required licenses when not immediately available.

Usage

```
set_license_queue_timeout [{value | unlimited | no_queue}]  
[-release_and_requeue_after release_and_requeue_after]
```

Description

Specifies the amount of time that the tool queues for one or more required licenses when not immediately available.

This command configures the time the tool waits for acquiring one or more required licenses (for example, during [set_context](#)). You can specify the same value during tool startup with the command line switch `-license_wait` (for example `tessent -license_wait`).

When the tool requires more than one license or if some licenses have alternatives, the tool queues for all of the possible licenses and acquires the first available. The tool reports progress every minute, showing which licenses are still missing and how long before timeout. If the tool cannot acquire a license during the specified time, the tool reports an error and aborts the operation that requested the license. You can interrupt the license queuing with Ctrl-C, which also aborts the operation that requested the license(s).

Arguments

- *value*
A positive integer that specifies the queuing time in minutes.
- **unlimited**
An optional string that specifies that there is no time limit and the tool will wait until a license is available. This is the default behavior.
- **no_queue**
An optional string that specifies that the tool not queue for a license and issue an error if not available immediately.
- **-release_and_requeue_after *release_and_requeue_after***
A switch and positive integer pair that specifies how many minutes the tool queues before releasing already-acquired licenses and performing a re-queue of all required licenses.

When you specify this option, the Tessent Shell tool's license queuing mechanism releases licenses the tool has acquired with every `-release_and_requeue_after` minutes before re-

requesting them. This allows other tools to acquire the licenses under the following circumstances:

- The tool is queuing for more than one license.
- The tool has acquired one license.
- The tool is queuing for one or more licenses that are currently in queue state.

Examples

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3  
SETUP> get_license_queue_timeout
```

3

Related Topics

[get_license_queue_timeout](#)

[tessent](#)

set_list_file

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the name of the list file into which the tool places the pins' logic values during simulation.

Usage

`set_list_file [filename] [-Replace]`

Description

Specifies the name of the list file into which the tool places the pins' logic values during simulation.

The `set_list_file` command specifies the file in which the tool places the simulation values for the pins which you previously identified with the `add_lists` command. The default behavior is for the tool to display the simulation values for the pins on standard output.

You can display the list of reported pins by using the `report_lists` command.

Arguments

- *filename*
A string that specifies the name of the file in which the tool places the logic values of pins during simulation. If you do not provide a *filename*, the default is standard output.
- `-Replace`
An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following Tessent FastScan example creates a file to store simulation values that are being reported:

```
set_system_mode analysis
add_lists /i_1006/o /i_1007/o
set_list_file listfile
create_patterns
```

Related Topics

[add_lists](#)
[delete_lists](#)
[report_lists](#)

set_logfile_handling

Context: unspecified, all contexts

Mode: all modes

Specifies for the tool to direct the transcript information to a file.

Usage

```
set_logfile_handling [-RESume | {filename [-Replace | -Append]}]
```

Description

Specifies for the tool to direct the transcript information to a file.

The set_logfile_handling command causes the tool to write the transcript information, which includes the commands and the corresponding output (if any), into the file you specify. You can execute the set_logfile_handling command at any time, as many times as you need.

In the logfile, the **command** keyword (or **sub-command** when a tool command is embedded in a Tcl command) precedes all commands that the tool executes. You can easily search for the executed commands, generate a separate dofile containing those commands, and then execute the dofile, thereby rerunning those commands within the tool.

When you set the logfile handling, the tool still writes the same information to the session transcript window in addition to the logfile. However, you can disable the writing of the information to the transcript window with the set_screen_display command.

If you want to stop writing to a logfile, issue the set_logfile_handling command with no options, which closes the appropriate file.

Arguments

- **-Resume**

An optional switch that specifies for the tool to resume writing transcript output to the logfile specified at invocation with the -Logfile invocation switch. You do not need to include the name of the invocation logfile with this switch; the tool will remember the name.

- **filename**

A string that specifies the name of the file to which you want the tool to write the transcript output. This string can be a full pathname or a leafname. If you only specify a leafname, the tool creates the file in the directory from which you invoked the tool.

If you do not specify a *filename*, the tool discontinues writing to the logfile and closes it.

- **-Replace**

An optional switch that forces the tool to overwrite the file if a file by that name already exists.

- **-Append**

An optional switch that causes the tool to begin writing the transcript at the end of the specified file.

Examples

The following example writes a logfile and disables the display of the transcript information in the transcript window:

```
set_logfile_handling /user/designs/setup_logfile
set_screen_display off
add_clocks 0 clk
add_clocks 1 pre clr
report_clocks
```

The following information shows what the logfile contains after running the preceding set of commands:

```
// command: set_screen_display off
// command: add_clocks 0 clk
// command: add_clocks 1 pre clr
// command: report_clocks
User-defined Clocks (2):
=====
Sync and Async Source Clocks
=====
-----  -----  -----  -----
Name    Off State Constraints Internal
-----  -----  -----  -----
'PRE'      1          No
'CLR'      1          No
'CLK/'     0          No
```

Related Topics

[get_logfile](#)
[report_environment](#)
[set_transcript_style](#)
[set_screen_display](#)
[tesson](#)

set_logical_design_libraries

Context: all contexts

Mode: setup

Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.

Usage

```
set_logical_design_libraries -logical_library_list logical_library_list
    [-default_library default_library_name] [-force]
```

Description

Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.

You must use this command before any design modules have been read in unless you specify the -force switch. If you specify -force, all design data that was read before is lost and must be re-read or recreated.

In the no_rtl context, this command is not allowed; a single library called “work” is used.

As you can see in the example, you can create a Tcl list of library names and use it as the *logical_library_list* value. You can sort this Tcl list from the bottom up to order the VHDL libraries into an order such that the libraries that depend on other libraries are defined after those they depend on. The default library is the last item in the list unless the -default_library is used to specify a different one.

Arguments

- **-logical_library_list** *logical_library_list*
A required switch and string pair that specifies the list of logical libraries.
- **-default_library** *default_library_name*
An optional switch and string pair that specifies the library name to be used as the default library. If this switch is not specified, the last library name in the *logical_library_list* is used as the default library.

Except for the get_modules command, all commands that accept a module name as one of their options look for the module name inside the default library. For example, to create an instance of module M1 from library L1, you use the following command:

```
create_instance instance_name -of_module [get_module M1 -
    in_library L1]
```

Using -of_module M1 creates an instance of module M1 from the default library.

- **-force**

An optional switch that suppresses the error message normally displayed when this command is executed after modules have already been read in or created. When the -force option is used, all pre-existing modules are deleted.

Examples

In this example, a Tcl list is defined with all logical library names and is ordered with the libraries that depend on other libraries defined after those they depend on. A Tcl array called vhdl_files is used to specify which files goes in which library. A simple design-independent script at the bottom is used to load those VHDL files into the right libraries. A similar technique can be used for loading Verilog into multiple design libraries.

```
set vhdl_dir design/vhdl/src
set lib_list [list utility lib1 work]
set vhdl_files(utility) [list \
    ${vhdl_dir}/f1.vhd\
    ${vhdl_dir}/f2.vhd\
]
set vhdl_files(lib1) [list \
    ${vhdl_dir}/f3.vhd\
    ${vhdl_dir}/f4.vhd\
]
set vhdl_files(work) [list \
    ${vhdl_dir}/f5.vhd\
    ${vhdl_dir}/f6.vhd\
]
set_logical_design_libraries -logical_library_list $lib_list
foreach lib $lib_list {
    if {[array names vhdl_files -exact $lib] == $lib} {
        read_vhdl $vhdl_files($lib) -in_library $lib
    }
}
```

Related Topics

[get_logical_library_list](#)
[read_verilog](#)
[read_vhdl](#)
[set_current_design](#)

set_loop_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies how the tool handles feedback networks.

Usage

```
set_loop_handling [{Tiex [-Duplication {ON | OFF}]} | Simulation]  
[-MAX_Stored_loops {number | UNLIMITED}]
```

Description

Specifies how the tool handles feedback networks.

The set_loop_handling command lets you perform DRC simulation of circuits containing combinational feedback networks.

The set_loop_handling command specifies the tool's loop handling behavior by either inserting a TIE-X gate or by stabilizing the loop values through an iterative simulation process.

By using the Tiex setting, you have the option to use gate duplication to reduce the impact that a TIE-X gate places on the circuit to break combinational loops. By default, this duplication switch is off.

For an iterative simulation, the tool inserts FB_BUF gates to break the combinational loops. It also optimizes the number of iterations performed to both stabilize the circuit and minimize the impact of the iterative process on performance and memory usage.

For another look at combinational feedback loops, refer to “[Feedback Loops](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- Tiex
A literal that specifies that TIE-X gates are used to break combinational loops.
- Simulation
A literal that specifies for the tool to use a simulation process to stabilize values in the loop. This option gives more accurate simulation results than other options. This is the default.
- -Duplication ON | OFF
An optional switch and literal pair that specify whether the tool can insert duplicate gates to reduce the impact of the gates that the tool places to break combinational loops. The literal choices are as follows:
 - ON — An optional literal that specifies for the circuit learning process to generate duplicate gates within any identified feedback paths.

- **OFF**— An optional literal that specifies for the circuit learning process to not generate duplicate gates within any identified feedback paths. This is the default upon invocation.
- **-MAX_Stored_loops {number | UNLIMITED}**
 An optional switch and integer or keyword pair that specifies the maximum number of loops that can be stored for reporting (using the `report_loops` command). By default, the tool stores up to 1000 loops. Use this switch if you want to store a specific number or all loops for reporting after the flattening process.

Examples

Example 1

The following Tessent FastScan example inserts a TIE-X gate to break any identified combinational or sequential asynchronous loop, then performs ATPG:

```
set_loop_handling tieX
set_system_mode analysis
create_patterns
```

Example 2

The following example shows how the tool indicates a design with more than 1000 combinational loops during the flattening process, and how to store all loops and write the loops to file loops.txt.

```
create_flat_model
// Note: Number of combinational loops exceeds storage limit (1000).
// Use 'set_loop_handling -max_stored_loops' to modify the number of loops stored and
// reported by 'report_loops'.
// Flattening process completed, cell instances=28234, gates=45400, PIs=110, POs=103,
// CPU time=93.00 sec.

// -----
// Begin circuit learning analyses.
// -----
// Equivalent gates=54  classes=0  CPU time=9.00 sec.

set_loop_handling -max_stored_loops unlimited
// Warning: Flat model deleted.

create_flat_model
// Flattening process completed, cell instances=28234, gates=45400, PIs=110,
// POs=103, CPU time=93.00 sec.

// -----
// Begin circuit learning analyses.
// -----
// Equivalent gates=54  classes=0  CPU time=9.00 sec.

report loops > loops.txt
```

Related Topics

[set_tla_loop_handling](#)

[report_loops](#)

set_lpct_condition_bits

Context: dft -edt

Mode: setup

Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.

Usage

```
set_lpct_condition_bits {-Condition {Reset | Scan_enable}  
[pin_name -on_controller_module] {-From connection_name | -No_connection}}
```

Description

Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.

The condition scan cells (condition bits) must already exist in the design core. If no condition bits are specified, the LPCT controller fully controls the reset and scan_enable signals. When LPCT condition bits are specified, the appropriate control bits are added to the test configuration data register in the LPCT controller. For each scan enable condition and each reset condition, one bit is added to the test configuration data register.

During ATPG, the scan cell specified using set_lpct_condition_bits *-from connection_name* is justified to the correct value to ensure the correct value of reset and scan enable.

This command is used in the IP Creation phase; it can only be used to create the Type 3 LPCT controller. For more information, see the Type 3 Controller in “[LPCT Controller Types](#)” in the *TestKompress User’s Manual*.

Arguments

- **-Condition {Reset | Scan_enable}**

Required switch and literal pair that specifies the condition bit type and the condition scan cell pin pathname. Literal options include:

Reset — Condition bit for controlling design reset during capture.

Scan_enable — Condition bit for controlling design scan enable during capture.

- **pin_name -on_controller_module**

Optional string and switch pair that specifies the pin name in the controller RTL for that particular input; this does not specify the connection.

- **-From connection_name | -No_connection**

Optional switch and string or switch that specifies the pin corresponding to the condition bits. This argument specifies the input connection to the LPCT controller. The *-no_connection* switch removes a previously specified condition.

Examples

The following example defines condition bits in the design to control the reset and scan_enable signals for the LPCT controller:

```
// Condition scan cell outputs (will be connected to LPCT logic)
set_lpct_condition_bits -condition reset -from c1/Q
set_lpct_condition_bits -condition scan_en -from c2/Q
```

Related Topics

[report_lpct_configuration](#)

[report_lpct_pins](#)

[set_lpct_controller](#)

[set_lpct_instances](#)

[set_lpct_pins](#)

set_lpct_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.

Usage

```
set_lpct_controller OFF |On [-MAX_SHift_cycles shift_cycles] [-MAX_Capture_cycles  
    capture_cycles] [-MAX_Scan_patterns pattern_count] [-MAX_Chain_patterns  
    pattern_count] [-TEst_mode_detect {signal | sequence binary_seq count}]  
    [-SHIFT_Control {Enable | Clock | None}] [-LOAD_unload_cycles count1 count2]  
    [-TAp_controller_interface {Off | ON}] [-Generate_scan_enable {Off | ON}]  
    [-TESSENT_Occ {Auto | ON}] [-TESSENT_OCC_Capture_cycle_width {1 | 2 | 3}]
```

Description

Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.

This command is used during EDT logic creation to create the necessary LPCT controller logic for a test application and during pattern generation to verify the compatibility between the LPCT controller configuration and the design core. This command is also carried forward into the dofile output for ATPG.

Arguments

- **OFF**

Required literal that disables the creation of an LPCT controller. Default setting.

- **On**

Required literal that enables the creation of an LPCT controller.

- **-MAX_SHift_cycles shift_cycles**—Required switch and integer pair that specifies the number of shift cycles in the longest scan chain. This number can be an estimate but it should take into consideration both the EDT and bypass modes. This number affects the size of the shift cycle counter.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- **-MAX_Capture_cycles capture_cycles**—Required switch and integer pair that specifies the maximum number of capture cycles for all test patterns. This number should take into consideration all of the types of test patterns generated for the test application.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- **-MAX_Scan_patterns *pattern_count***

Optional switch and integer pair that specifies the maximum number of scan test patterns generated for any test pattern type in the test application. You must consider all the test pattern types used when determining this number.

This number determines the size of the scan test pattern counter created for the LPCT controller. By default, a counter of size 20 bits is used which implies a maximum of 1048576 scan test patterns.

- **-MAX_Chain_patterns *pattern_count***

Optional switch that specifies the maximum number of chain test patterns generated for any test pattern type in your test application. You must consider the maximum number of test patterns generated for any chain test type when determining this number. The default size of the chain test pattern counter is 10 bits which implies no more than 1024 chain test patterns.

- **-TEst_mode_detect {*signal / sequence binary_seq count*}**

Optional switch and literal that configures how the LPCT controller enters test mode. By default, test mode is enabled when the *test_mode* signal is asserted.

signal — The Set LPCT Pin *test_mode* command specifies the name of the signal to be used to enable test mode.

sequence binary_seq count — The LPCT controller enables test mode if the specified binary sequence, specified by *binary_seq*, is detected within the first *count* cycles after reset.

- **-SHIFT_Control {Enable | Clock | None}**

Optional switch and literal pair that specifies how the LPCT controller generates the shift clock control signal. Options include:

Enable — A *lpct_shift_en* enable signal is output to generate the shift clock waveform. This is the default setting.

Clock — A *lpct_shift_clock* signal is generated when an internal clock is pulsed during shift. Note, if all shift clocks are top-level clocks, they are controllable from top-pins during both shift and capture cycles; in this case, an additional *lpct_shift_clock* is not needed and is therefore not generated.

None — No output signal. This option is used when clocks are available at the top-level.

- **-LOAD_unload_cycles *count1 count2***

Optional switch and double integer that specifies the number of cycles that precede and succeed the shift cycles in the *load_unload* procedure. Empty cycles corresponding to the specified values are added to the test procedure. This argument must be specified during IP creation.

These values are determined by the number of cycles before and after the *apply shift* in the *load_unload* procedure. Specified values should be higher than the number of cycles in the input test procedure, otherwise they are ignored.

By default, the Type 2 and Type 3 controllers have two pre-shift and two post_shift cycles. By default, the Type 1 controller has zero pre-shift and two post-shift cycles.

- **-TAp_controller_interface {Off | ON}**

Optional switch and literal that specifies to use a TAP-generated scan enable version of the controller.

- **-Generate_scan_enable {Off | ON}**

Optional switch and literal that specifies whether the scan_en signal is generated internally by the LPCT controller.

Off — If the -TAp_controller_interface argument is set to Off, the LPCT controller has a top-level scan enable signal. This is the default configuration. If the -TAp_controller_interface argument is set to On, an error is issued.

ON — If the -TAp_controller_interface argument is set to Off, the scan enable signal is internally generated by the LPCT controller. If the -TAp_controller_interface argument is set to On, the LPCT the scan enable is generated using the TAP state machine.

- **-TESSENT_Occ {Auto | ON}**

An optional switch and literal that generates a Tessent OCC-compatible LPCT controller for skeleton designs that do not contain Tessent OCC instances. You specify this switch during pre-synthesis IP creation. Choose one of the following options:

auto — Infers the Tessent OCC instruments from the TCD and the ICL. This is the default.

on — Enables generation of Tessent OCC compatible LPCT controller when OCC core instances are not added.

- **-TESSENT_OCC_Capture_cycle_width {1 | 2 | 3}**

An optional switch and literal that specifies how many bits are present in the OCC, this must be a number between 1 and 3 (corresponding to capture_window_size 2 through 6 in the OCC DftSpecification). The default is 2 specifies the capture cycle width.

Examples

The following example enables an LPCT controller and configures it to use a sequence to enter test mode:

```
//define parameters for test controller
set_lpct_controller on -max_shift_cycles 400 -max_capture_cycles 2
set_lpct_controller -test_mode_detect sequence 1100100101001110 64
```

Related Topics

[report_lpct_configuration](#)

[report_lpct_pins](#)

[set_lpct_instances](#)

set_lpct_instances

Context: dft -edt

Mode: setup

Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.

Usage

```
set_lpct_instances -Block_location instance_pathname
                   -Clock_gater_location instance_pathname
```

Description

Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.

The clock gater is added only if [set_lpct_controller -shift_control_clock](#) is specified.

Arguments

- **-Block_location *instance_pathname***
Optional switch and string pair that specifies an instance in the design core in which to insert the LPCT controller. By default, the LPCT controller is inserted at the top-level of the design core.
- **-Clock_gater_location *instance_pathname***
Optional switch and string pair that specifies an instance in which to insert the LPCT clock gater. By default, the clock gater is inserted at the top-level of the design core.

Examples

The following example places the LPCT controller inside the *my_design/xyz/foo* instance in the design core:

```
//define parameters for LPCT controller

set_lpct_controller on -max_shift_cycles 5000 -max_capture_cycles 4 -max_chain_patterns
1000
set_lpct_controller -shift_control_clock
set_lpct_instances -block_location my_design/xyz/foo
```

Related Topics

[report_lpct_configuration](#)
[report_lpct_pins](#)
[set_lpct_controller](#)

set_lpct_pins

Context: dft -edt

Mode: setup

Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.

Usage

Specify for all configuration

```
set_lpct_pins
  [Clock pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]] |
  [CLOCK_Mux_Select [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
  [CAPTURE_Enable [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
  [SHIFT_Enable [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
  [SHIFT_Clock [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
```

Specify for Type 1 LPCT controllers (external scan_en)

```
set_lpct_pins
  [INPUT_Scan_enable pin_name [internal_node_name]
    [-ON_controller_module] [-NOInv | -INV]]
  [TEST_Clock_connection [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
```

Specify for Type 2 or Type 3 LPCT controllers (internally-generated scan_en TAP or LPCT generated)

```
set_lpct_pins
  [Reset pin_name [internal_node_name] [-ON_controller_module]
    [-Active High | Low] [-NOInv | -INV]]
  [OUTPUT_Scan_enable [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
```

Specify for Type 3 LPCT controller (scan_en generated by LPCT controller only)

```
set_lpct_pins
  [DATA_In pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]]
  [TEST_Mode pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]]
  [RESET_Out [pin_name -ON_controller_module] {connection_name | -No_connection}]
  [TEST_End [pin_name -ON_controller_module] {connection_name | -No_connection}]
  [TEST_Active [pin_name -ON_controller_module]
    {connection_name | -No_connection}]
```

Specify for Type 2 LPCT controller (scan_en generated using TAP state machine only)

```
set_lpct_pins
  [SHIFT_Dr pin_name [internal_node_name] [-ON_controller_module]
   [-NOInv | -INV]]
  [CAPTURE_Dr pin_name [internal_node_name] [-ON_controller_module]
   [-NOInv | -INV]]
  [Update_dr pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]
   [-ON_controller_module] [-NOInv | -INV]]
  [TMs pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]]
  [TEST_Clock_connection [pin_name -ON_controller_module]
   {connection_name | -No_connection}]
```

Description

Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.

When using the LPCT controller in which scan_en is generated using TAP states, the clock and reset inputs of the controller should be connected to TCK and the test_logic_reset pins on the TAP controller.

For hybrid TK/LBIST IP generation, the dft -edt -lbist context supports this command.

Arguments

- **Clock**

Required argument that specifies as always-pulse clock input for a flip-flop based design.

This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- *pin_name*

Optional string and switch pair that specify the pin name in the controller RTL for that particular input; this does not specify the connection. Depending on the application, it may be a top-level pin, an input pin, or an output pin as follows. Use this string with the *-on_controller_module* switch to name the output pin when scan cell conditions are used to reset the LPCT controller.

- *internal_node_name*

Optional string used together with the *pin_name* argument to specify the top-level pin name and corresponding internal connection for controller input pins. You can use a literal dash “-” for the *pin_name* argument when no top-level pin exists in the design.

- **-ON_controller_module**

Optional switch used with the *pin_name* argument that specifies the pin name in the controller RTL for each input; this does not specify the connection.

- *connection_name*
Optional switch used to specify the names of one or more internal pins to connect to the controller output pins.
- **-No_connection**
Optional switch used to specify that no connection is required for a controller output pin.
- **-NOInv| -INV**
Optional switch that specifies a LPCT signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.
You should specify **-INV** when there is an inversion between the internal connection name for the LPCT pin and the chip boundary such as when specifying an inverting pad.
Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.
- **CLOCK_Mux_select**
Optional argument that specifies the connection to be made to the *clock_mux_select* output of the controller. This pin is a MUX select signal that selects between the shift clock generated by the LPCT controller and the capture clock. If *clock_mux_select* is high, the shift clock is selected; if *clock_mux_select* is low, the capture clock is selected.
This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.
- **CAPTURE_Enable**
Optional argument that specifies the controller output signal that indicates when the controller is in capture mode. This option specifies the pin to which this *capture_enable* signal is to be connected.
This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.
- **SHIFT_Enable**
Optional argument that specifies a controller output signal that indicates when the controller is in shift. The *shift_enable* option specifies the pins to which the *shift_enable* signal should be connected. If you provide clock gating using *set_lpct_controller -shift_control enable*, this option is required; it is used as the clock gater enable signal to produce the shift clock from a pulse-always clock.
This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.
- **SHIFT_Clock**
Argument that specifies the pin is a shift clock signal.

This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- INPUT_Scan_enable

Argument that specifies the top-level scan enable port.

This is used as an input only to the Type 1 LPCT controller. This option can only be used for the Type 1 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- Reset

Required argument that specifies the reset input signal for the controller. The reset pin can be shared with the design’s functional reset. This can be either an external pin or internal power-on-reset.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- OUTPUT_Scan_enable

Argument that specifies the list of internal pins to which the LPCT output scan enable pin is connected.

This option is required when generating a Type 2 or Type 3 LPCT controller.

- DATA_In

Optional argument that specifies the pin is a data input to the test controller. For the Type 3 controller, a test sequence and configuration is scanned in through this port. This port is usually shared with the edt_channel input.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- TEST_Mode

Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- RESET_Out

Optional argument that specifies the reset signal to the design. This signal is generated by combining the functional reset signal (specified using “`set_lpct_pins reset`”) with the reset condition or reset control configuration inside the controller. This allows the tool to detect faults connected to the reset signal even while the top-level reset pin is constrained to OFF during capture. If this option is not specified, all the pins driven by the reset signal in the input netlist are changed to be driven by the `lpct_reset_out` output of the controller. If this option is specified, only the pins explicitly specified are driven by the `lpct_reset_out` output of the controller.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- TEST_End

Optional argument that specifies the pin is an end of test signal.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- TEST_Active

Optional argument that specifies an output signal from that LPCT controller that indicates the LPCT controller is active and controlling the scan test.

This option is only valid for the Type 3 controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- SHIFT_Dr

Required argument that specifies the pin is a Shift_Dr signal on the TAP controller.

This option can only be used for the Type 2 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- CAPTURE_Dr

Required argument that specifies the pin is a Capture_Dr signal on the TAP controller.

This option can only be used for the Type 2 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- Update_dr

Required argument that specifies the pin is an Update_Dr signal on the TAP controller.

This option can only be used for the Type 2 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- TMs

Required argument that specifies the pin is a test mode select tms input pin of the TAP.

This option can only be used for the Type 2 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- -Active {High|Low}

Optional switch and literal pair that specifies the active state for the input reset signal. By default, the reset is active high.

This option can only be used for the Type 3 LPCT controller. For more information, see “[LPCT Controller Types](#)” in the *Tessent TestKompress User’s Manual*.

- TEST_Clock_connection

Optional switch that specifies the test clock input pin for the user-added test_mode mux. The tool connects the output of the clock gater, added to facilitate sharing of the LPCT clock

and the scan clock, to the specified pin. The user must connect the other input of the mux to the functional clock and the select pin to the test mode signal.

Examples

Example 1

The following example defines the connections between the LPCT controller and the design core.

```
// Temporary top-level scan IO pins (removed by TK during IP creation

add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains chain2 grp1 scan_in2 scan_out2

// LPCT controller input pins

set_lpct_pins clock ref_clk clk_pad/Z
set_lpct_pins reset_out - pwr_up/rst
set_lpct_pins data_in fnl_input

// Connections from LPCT logic outputs to design

set_lpct_pins output_scan_enable scan_en
set_lpct_controller -shift_control enable
set_lpct_pins shift_en clk_mux/shift_en
set_lpct_pins capture_en clk_mux/cap_en
```

Example 2

The following example defines the connections between the LPCT controller and the design core when the LPCT controller internally generates the scan_en signal.

```
set_lpct_controller -generate_scan_enable on -max_shift 1000 -max_capture 3
set_lpct_pins output_scan_enable scan_mode
```

Example 3

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists.

```
set_lpct_controller on
set_lpct_pins input_scan_enable scan_en
```

Example 4

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists and the user has implemented clock control.

```
set_lpct_controller -shift_control enable
set_lpct_pins capture_en pll/capture_start
set_lpct_pins clock_mux_select pll/select
set_lpct_pins shift_en pll/shift_en
```

Example 5

The following example defines the connections between the LPCT controller and the design core when a TAP generated scan_en signal exists.

```
set_lpct_controller -tap_controller on -generate_scan_enable on
set_lpct_pins output_scan_enable scan_mode
set_lpct_pins shift_dr - tap/shiftEn
set_lpct_pins capture_dr - tap/captureEn
set_lpct_pins update_dr - tap/updateEn
set_lpct_pins test_mode - tap/atpgInstruction
set_lpct_pins reset - tap/resetl -active low
```

Example 6

The following example defines the connections between the LPCT controller and design core when a TAP is used to generate scan_en signal

```
set_lpct_pins clock refclk -on_controller_module      // input pin
set_lpct_pins capture_en pll_start -on_controller_module // output pin
```

Example 7

The following example specifies to use a top-level scan clock as the LPCT clock for a Type 1 controller.

```
set_context dft -edt
add_clocks 0 clk           // There is a single clock in the design
add_scan_chains ...
set_lpct_controller on -shift_control_clock
set_lpct_pin clock clk     // LPCT clock is shared with scan clock
set_lpct_pin input_scan_enable scan_en
set_lpct_pin test_clock_connection test_mode_mux/B
set_system_mode analysis
report_lpct_pins
write_edt_files created -replace
```

Example 8

The following example specifies to use tck as the scan shift clock in a Type 2 LPCT controller. Note, the test mode mux that selects between the LPCT-generated scan clock and the functional clock already exists in the design. The test clock connection pin on the mux is specified with the test_clock_connection pin type.

```
set_context dft -edt
add_clocks 0 tck
add_scan_chains ...
set_lpct_controller -tap_controller_interface on
set_lpct_controller -shift_control clock
set_lpct_pins output_scan_enable scan_en
set_lpct_pins clock tck pad_tck/Z // LPCT clock is tck
set_lpct_pins tms tms pad_tms/Z
set_lpct_pins reset - tap_i/tlr
set_lpct_pins capture_dr - tap_i/capturedr
set_lpct_pins shift_dr - tap_i/shiftdr
set_lpct_pins update_dr - tap_i/updatedr
set_lpct_pins test_mode - tap_i/edt_scan_inst
set_lpct_pins test_clock_connection test_mode_mux/B
set_edt_options -channel 1
set_edt_pins input 1 tdi pad_tdi/Z
set_edt_pins output 1 tdo tap_i/tap_edt_channel_reg_in
set_system_mode analysis
report_lpct_pins
report_lpct_configuration
write_edt_files created -replace
```

Related Topics

[report_lpct_configuration](#)

[report_lpct_pins](#)

[set_lpct_controller](#)

[set_macrotest_options](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Prepares a black box macro to allow MacroTest conversion of patterns at the boundaries of that instance.

Usage

`set_macrotest_options {OFF | ON}`

Description

Prepares a black box macro to allow MacroTest conversion of patterns at the boundaries of that instance.

The `set_macrotest_options` command changes two normal behaviors of the tool's DRC. To obtain a pulse from a macro, a PI of the circuit that has been declared as a clock, read control, or write control must drive the pin of the macro. If the black box is an empty instantiation of a RAM, it is likely that the tool needs to pulse the write control to write into the RAM. Normally, if a write control does not connect to a `_ram` or `_cram` primitive, DRC issues a G5 violation and does not take you out of Setup until the write control is deleted. The `set_macrotest_options` command prevents this and treats any write control as valid (as if it were connected to a `_ram` or `_cram`).

Another DRC aspect that the tool modifies occurs if a transparent latch (TLA) is on the control (input) side of a black box macro, and is unobservable due to the macro. Tessent FastScan converts a black box macro to a set of TieX gates, one per output. As a result, if the TLA only fans out to the macro, the tool cannot observe it (it has no sensitizable path to some PO or scan cell). Normally, DRC converts such a TLA to a TieX gate, preventing MacroTest from justifying values on the input side of the black box macro backward through the TLA. This is because it thinks the TieX means the primitive cannot be controlled. The `set_macrotest_options` command prevents this and keeps unobservable TLAs as TLAs.

If in doubt, issue the `set_macrotest_options` command in setup mode before proceeding with macro testing. No errors occur if DRC does not encounter the situations resolved by this command. Therefore, you can issue the `set_macrotest_options` command before every MacroTest run if you desire.

Arguments

- `{OFF | ON}`

A required Boolean that specifies whether to modify DRC as described in the preceding description.

OFF—Disables DRC modification (normal DRC is performed). This is the default at invocation.

ON — Enables modified DRC analysis to avoid potential macrotest-specific ATPG issues.

Examples

The following example shows the command issued to allow black box macro testing:

```
set_macrotest_options on
```

Related Topics

[macrotest](#)

set_memory_identification_options

Context: dft

Mode: setup

Sets options and pattern expressions to validate memory TCD are associated with memory modules to be tested by MemoryBIST.

Usage

```
set_memory_identification_options {name_patterns [-regexp] [-nocase]} | [-off]
```

Description

Sets regular expression or wildcard naming patterns to identify memory module names in the design technology library. When `check_design_rules` is run, Tessent MemoryBIST will use this information to identify the memories instantiated in the design that have not been declared as memories and will not be tested by MemoryBIST. This condition can be caused by a missing memory TCD for a design's memory module. When a memory library file is read in, the tool will automatically associate the memory TCD to the design module and will declare it as a memory to be tested by MemoryBIST.

The options specific to the `set_memory_identification_options` command are used by the `check_design_rules` command to ensure that all modules that should be declared as memories are properly recognized after a successful system mode transition. An error is issued if a design module matches the provided *name_patterns* argument, but no associated memory TCD is found. A warning is issued if there are design modules recognized as memories, but they do not match the *name_patterns* argument. In cases where errors or warnings are issued, the tool automatically displays a tabular report from the validation. The validation report can always be displayed with the [report_memory_identification](#) command.

Arguments

- ***name_patterns***

A required string or Tcl list of strings that specifies one or more patterns to be used to identify memories in the design. A non-empty string must be provided or an error is issued.

- **-regexp**

An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of a simple wildcard pattern. Refer to the “[Glob and Regular Expression Pattern Matching Syntax](#)” section for a description of the regular expression syntax.

- **-nocase**

An optional switch that directs the tool to perform case-insensitive pattern matching when searching for modules matching the *name_patterns*.

- -off

An optional switch that is mutually exclusive with all other arguments. Specifying the -off switch eliminates module matching validation that was previously enabled with a set_memory_identification_options command executed with a valid *name_patterns* expression.

Examples

Assume the following memory modules exist in a design and it is desired to identify them as memory modules:

```
SYNC_1R1W_16x8, SYNC_1R1W_32x4, RAM_32x4, RAM_128x128
```

The following use of set_memory_identification_options can be used to match these memory modules:

```
set_memory_identification_options {SYNC.* RAM.*} -regexp
```

Related Topics

[report_memory_identification](#)

set_memory_instance_options

Context: dft

Mode: setup

Prerequisite: The current design must be set with the [set_current_design](#) command.

Set options on or for memory instances to be used by the [check_design_rules](#) and [create_dft_specification](#) commands.

Usage

```
set_memory_instance_options {memory_instances}  
  [-bist_data_in_pipelining on | off | int] [-physical_cluster_override string]  
  [-test_clock_override domain_label]  
  [-repair_sharing off | on | auto]  
  [-use_in_memory_bist_dft_specification off | on | auto]  
  [-use_in_memory_bisr_dft_specification off | on | auto]} |  
  [-bisr_segment_order_file file_path] |  
  [-physical_cluster_size_ratio int]
```

Description

Sets options on or for memory instances to be used by the [check_design_rules](#) and [create_dft_specification](#) command, when the [set_dft_specification_requirements](#) command was used to enable the memory BIST and or memory BISR feature.

This command is used to set options on memory instances as returned by the [get_memory_instances](#) command. Most options can be set differently for each memory instance. The `-bisr_segment_order_file` and the `-physical_cluster_size_ratio` options apply to all instances.

Arguments

- ***memory_instances***

A string that contains a Tcl list of one or more names of memory instances, or a collection of one or more memory instances. The memory instances are the instances on which the `tcd_memory_name` [Instance](#) attribute is defined as a result of a TCD Memory description having been matched to its module name—see “[Tessent Core Description](#)” on page 3755. You can use the [get_memory_instances](#) command to get a collection of all memory instances.

- **-bist_data_in_pipelining on | off | *int***

An optional switch and value pair that specifies the number of data-in pipelining stages the specific memories are to use during memory BIST. It is specified here as it affects the memory BIST partitioning because only memories with equal data-in pipelining can be tested in the same controller step. The value off is equivalent to 0 and the value on is equivalent to 1.

- **-physical_cluster_override *string***

An optional switch and value pair that associates one or more memory instances to a given cluster ID. The string is allowed to contain any number of letters, numbers, and underscores; it cannot start with the word “def” and be followed by an integer because those strings are reserved for the clusters computed from the DEF file. If you read a DEF file using the [read_def](#) command, physical clusters will be created for all memory instances for which a physical cluster override string was not specified. See the description of the **-physical_cluster_size_ratio** option below for more information about the clustering process. The automatic cluster generation computed from DEF is performed when running the [check_design_rules](#) commands.

- **-test_clock_override *domain_label***

An optional switch and value pair that associates one or more memory instances to a clock domain label. By default, the clock domain label associated to the memory is one of the clock domains connected to the clock ports of the memory. When the clocks have different frequencies, one of the fastest ones is selected. You can use this option when the memory instance has no clocks (that is, asynchronous memory) or to force the selection of a given clock when more than one clock has the same frequency. Clock ports receiving a clock that is different than the selected clock domain will have a multiplexer inserted on it to inject a common clock during memory BIST.

- **-repair_sharing off | on | auto**

An optional switch and literal pair that specifies whether the memory instance will have memory repair sharing enabled. By default, the value of “auto” will use the global enable value specified in the **DefaultsSpecification/DftSpecification/MemoryBist/RepairOptions/repair_sharing** property. A value of “auto” will also use the global repair group scope specified by the **repair_group_scope** property in the **DefaultsSpecification/DftSpecification MemoryInterfaceOptions** or **MemoryClusterOptions** wrappers. Specifying “off” or “on” will override the global values specified in the **DefaultsSpecification** for the memory instance.

- **-use_in_memory_bist_dft_specification off | on | auto**

An optional switch and literal pair that specifies whether the memory instance is to be considered by DRC and as a part of the **DftSpecification/MemoryBist** wrapper that the **create_dft_specification** command will create. When set to auto, the memory instance is considered if it is not seen as already connected to a memory BIST interface module. If it is already connected to a memory BIST interface module, it is excluded from the new **DftSpecification/MemoryBist** wrapper.

This feature is useful when doing multi-pass insertion in the case where you initially added memory BIST for a set of memories and then, later, edited the RTL and inserted new memories. When you specify the default value of auto, re-running the memory BIST insertion process on this RTL will only consider the new memory. You can force the exclusion of a memory instance even if it is not already connected to a memory BIST interface module by specifying the off value.

- `-use_in_memory_bisr_dft_specification off | on | auto`

An optional switch and literal pair that specifies whether the memory instance is to be part of the [DftSpecification/MemoryBisr](#) wrapper that the [create_dft_specification](#) command will create. When set to auto, the memory instance is considered if it is not seen as already connected to a memory BISR register module. If it is already connected to a memory BISR register module, it is excluded from the new [DftSpecification/MemoryBisr](#) wrapper.

This feature is useful when doing multi-pass insertion in the case where you initially added memory BISR for a set of memories and then, later, edited the RTL and inserted new repairable memories. When you specify the default value of auto, re-running the memory BISR insertion process on this RTL will only consider the new repairable memory. You can force the exclusion of a memory instance even if it is not already connected to a memory BISR register module by specifying the off value.

- `-bisr_segment_order_file file_path`

An optional switch and value pair that references a [MemoryBisr](#) bisr_segment_order_file following the format described in [Figure 10-53](#) on page 3480. By default, a BISR segment order file is created when the [check_design_rules](#) command is run if repairable memories and/or child blocks with BISR chains are present in the current design. If you read a DEF file using the [read_def](#) command, this file is automatically created with an ordering extracted from the DEF file in order to minimize routing. If you did not provide a DEF file, you must order the BISR segment order file manually and then reference the file using this option. The specified BISR Segment Order File applies to all memories in the design that will be processed by the next [create_dft_specification](#) command.

This option cannot be combined with the `memory_instances` option or any of the memory instance-related options explained before.

- `-physical_cluster_size_ratio int`

An optional switch and value pair that specifies the parameter used to create memory clusters for Memory BIST partitioning. If you read a DEF file using the [read_def](#) command, memory clusters are created and memories not belonging to the same cluster are assigned to different memory BIST controllers when the [create_dft_specification](#) command creates the [DftSpecification/MemoryBist](#) wrapper.

A rectangle is created to fully enclose the current design and the length of its diagonal is extracted. The ratio is the percentage of a cluster's diameter with respect to this diagonal distance. A memory instance will be added to a given cluster, if it does not cause the diameter of the cluster to exceed the threshold ratio. Otherwise, a new cluster is created. The default ratio is 20% but you can change this percentage value by specifying an integer between 0 and 100 using the `-physical_cluster_size_ratio` option. A value of 0 causes each memory instance to be in its own cluster, and a value of 100 results in all memory instances being in the same (single) cluster.

Examples

Example 1

The following example sets the number of data in the pipelining stages to 3 for the memory instances found below instance u1/u2. It also sets the **-physical_cluster_size_ratio option to 30%**.

```
set_memory_instance_option [get_memory_instances -below_instance u1/u2] \
    -bist_data_in_pipelining 3
set_memory_instance_option -physical_cluster_size_ratio 30
```

Example 2

The following example defines the BISR segment order file that the tool will use for all memories in the design.

```
set_memory_instance_options -bisr_segment_order_file ./top.bisr_segment_order
```

Related Topics

[get_memory_instances](#)

[get_memory_instance_option](#)

[set_dft_specification_requirements](#)

[report_ijtag_instances](#)

set_misr_connections

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Describes the connections between the scan chain outputs and the MISR.

Usage

```
set_misr_connections misr_name -tap integer -chain chain_name
```

Description

Describes the connections between the scan chain outputs and the MISR.

This command is used with the hybrid TK/LBIST flow—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

If you map multiple scan chains to the same MISR position, then the tool infers a XOR spatial compactor that maps from those multiple scan chain outputs to the specified MISR feedback position.

Arguments

- ***misr_name***

A required string that specifies the name of the MISR specified with the add_misrs command.

- **-tap *integer***

A required switch and required and repeatable integer that specifies the tap points for the MISR feedback network. The tap index starts from 1, to be consistent with the decompressor commands.

- **-chain *chain_name***

A required switch and string that specifies the name of the scan chain pin that you want to connect to the MISR specified by ***misr_name***.

Examples

This example describes the hardware connection from scan chain outputs to MISR inputs through a 4-to-1 XOR compactor.

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5] -tap 2
```

set_misr_connections

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15] -tap 4
```

Related Topics

[add_misrs](#)

[delete_misrs](#)

set_module_matching_options

Context: all contexts

Mode: all modes

Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL module to a design module within ICL extraction.

Usage

```
set_module_matching_options [-prefix_pattern_list prefix_pattern_list]
                           [-suffix_pattern_list suffix_pattern_list] [-regexp] [-nocase] [-append]
```

Description

Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL or a TCD module to a design module. The module matching is performed in those situations as follows:

1. With a [set_current_design](#) command call. This forces the tool to perform a complete elaboration, which resets other settings such as already defined black boxes.
2. With a [set_context](#) command call, and the design is already elaborated.
3. With an [extract_icl](#) command call.

During module matching, the list of all design modules found below the current design is used to search for corresponding ICL and TCD modules. The ICL and TCD modules are then attached to the design modules.

If the ICL or TCD modules refer to the RTL names of the design modules and module matching is done on the synthesized view, the design modules that are used more than once are likely to have been unqualified with an integer suffix. You may also have instructed the synthesis tool to prefix every synthesized module with a common prefix. The default suffix list will handle the unification suffix inserted by synthesis automatically. If you instructed the synthesis tool to prefix every synthesized module with a common prefix, you need to describe the used prefix using the *-prefix_pattern_list* switch.

See the `instrument_name`, `tcd_bscan_name`, `tcd_fusebox_name`, `tcd_memory_cluster_name`, and `tcd_memory_name` [Instance](#) attribute descriptions. Those attributes are set on the design instances and modules when they are matched to an ICL or TCD module. See also the [get_ijtag_instances](#) and [get_memory_instances](#) command descriptions.

Arguments

- *-prefix_pattern_list *prefix_pattern_list**

An optional switch and string pair that specifies the prefix patterns taken into account when matching ICL modules to design modules. The *prefix_pattern_list* string or Tcl list of strings lists the prefix strings or regular expressions to be used as a prefix to the ICL module name when matching it against the design module names.

Note

 By default, the `set_insertion_options -edited_module_prefix` command and switch inserts an underscore character between the prefix and the original module name. When using the `-prefix_pattern_list` switch, you must explicitly specify this underscore.

- `-suffix_pattern_list suffix_pattern_list`

An optional switch and string pair that specifies the suffix strings or regular expressions to be used for unqualified design module mapping. This switch can only be used once. The `suffix_pattern_list` string or Tcl list of strings lists one or more patterns to be used for unqualified design module mapping. By default, the tool matches any module name with a suffix that matches the regular expression “`{[_]+[0-9]+}`”. This means that, by default, the tool will match all names of the format `<module_name>_<number>`.

- `-regexp`

An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section “[Glob and Regular Expression Pattern Matching Syntax](#)” on page 3164 for a description of the regular expression syntax.

- `-nocase`

An optional switch that forces the pattern matching to be case insensitive. The presence of the `-nocase` switch in the most recent `set_module_matching_options` command determines the behavior of the module matching. If the `-nocase` switch is used in the most recent `set_module_matching_options` command, then the pattern matching is case insensitive. If the `-nocase` switch is omitted, the case sensitivity depends on the hardware description language (Verilog or VHDL) that has been used to describe the design module.

- `-append`

An optional switch that specifies to append the newly specified pattern list to the existing list or to replace the existing list.

Return Values

None

Examples

In the following example, this command is used to allow matching ICL modules with an optional prefix "mycore_" and an optional suffix matching `_ts[0-9]+`. In this example, an ICL module InstrumentA will be allowed to match `mycore_InstrumentA_ts1` and `mycore_InstrumentA_ts2`.

```
set_module_matching_options -prefix_pattern_list {mycore_}
                           -suffix_pattern_list {_ts[0-9]+} -regexp
```

Related Topics

[report_module_matching](#)

[report_module_matching_options](#)

set_multiple_detection

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables the multiple detection of faults.

Usage

```
set_multiple_detection
  [-Guaranteed_atpg_detections guaranteed_atpg_detections]
  [-Desired_atpg_detections desired_atpg_detections]
  [-Simulation_drop_limit simulation_drop_limit]
  [-REPort_incremental_statistics {ON | OFF}] [OFF]
```

UDFM Usage:

```
set_multiple_detection
  [-Simulation_drop_limit simulation_drop_limit] [OFF]
```

Description

Enables the multiple detection of faults.

Multiple detection is a statistical way to gain bridge fault coverage. By default, the tool generates a single detection pattern set, where a fault is detected once, then dropped from the simulation.

When multiple detection is enabled, the DS fault category represents faults detected one or more times, not just faults detected the threshold number of times. The [create_patterns](#) command includes a cumulative Bridge Coverage Estimate (BCE) in the statistics transcription for each batch of patterns. Use the *simulation_drop_limit* argument to increase the accuracy of the BCE, being aware that the improved accuracy entails more run time.

The BCE, which is based on simulation results, is calculated as follows:

$$\text{BCE} = \left[\sum_{i=1}^n \frac{f_i}{TE} \cdot (1 - 2^{-i}) \right] + \frac{N_{DI}}{TE}$$

where:

f_i = number of faults detected i times by the test pattern set

TE = number of total testable faults in the target fault list

N_{DI} = number of DI faults in the target fault list
 $n = atpg_detections$ (or
 $fault_drop_detections$ if used)

Enabling multiple detections changes the behavior of the following commands:

- [analyze_compression](#) — If the `-Preserve_faults_from_analysis` switch is specified, preserves the N-detection information for all the detected faults during the `analyze_compression` operation.
- [create_patterns](#) — Performs multiple iterations (loops) as needed to ensure every testable fault is detected *guaranteed_atpg_detections* times and includes a cumulative bridge coverage estimate (BCE) in the statistics transcription for each batch of patterns simulated. Each pattern counts at most only one detection for any particular fault.
- [report_statistics](#) — Lists the final BCE in reports and additionally includes a histogram that shows the profile of the number of detections for multiple detection DS faults.
- [report_environment](#) — Lists a “multiple detection = ...” line that shows the current multiple detection settings. If this line is absent, it means multiple detection is disabled.
- [report_faults](#) — Displays a “num_detections...” line for each DS fault that shows the number of detections for that fault.
- [write_faults](#) — Writes a “num_detections...” line for each multiple detection DS fault that shows the number of detections for that fault.

Issue this command without any arguments to report the current setting. If you issue this command with arguments more than once, the last command overwrites the previous settings.

Note that when the fault type is UDFM, the `set_multiple_detection` command works only for simulation and not for ATPG.

Arguments

- `-Guaranteed_atpg_detections` *guaranteed_atpg_detections*

An optional switch and integer pair that specifies the number of detections required for each testable fault. The *guaranteed_atpg_detections* can be any integer between 1 and 255 and determines the number of detections for each fault that the [create_patterns](#) command should target during pattern generation. The specified number of detections for each fault are guaranteed during fault simulation before the tool drops the fault from the fault list.

The default value of *guaranteed_atpg_detections* is 1. Increasing the number of guaranteed detections can significantly increase the pattern count and ATPG run time.

- `-Desired_atpg_detections` *desired_atpg_detections*

An optional switch and integer pair that specifies the number of detections desired for each testable fault. These faults are detected using Embedded Multi-Detect (EMD) functionality. The *desired_atpg_detections* can be any integer between 1 and 255.

Setting *desired_atpg_detections* to 1 disables EMD.

By default, *desired_atpg_detections* is equal to *guaranteed_atpg_detections*.

The *desired_atpg_detections* cannot be less than the *guaranteed_atpg_detections* except when EMD is disabled (*desired_atpg_detections* set to 1).

Unlike the number of guaranteed detections, increasing the number of desired detections does not increase the pattern count. However, it may increase the ATPG run time. For optimal performance, it is not recommended to set the *desired_atpg_detections* to greater than 5.

- **-Simulation_drop_limit simulation_drop_limit**

An optional switch and integer pair that specifies the maximum number of detections for a fault. The tool drops the fault from the simulation after the *simulation_drop_limit*.

When the fault type is UDFM, you can set the *simulation_drop_limit* to any positive integer. By default, the *simulation_drop_limit* is 1.

For all other fault types, the *simulation_drop_limit* cannot be less than the *guaranteed_atpg_detections* or *desired_atpg_detections* values. If you do not specify a value, the *simulation_drop_limit* is automatically set to one of the following:

- **guaranteed_atpg_detections = desired_atpg_detections = 1**

The *simulation_drop_limit* is set to 1.

- **guaranteed_atpg_detections <= 10 AND 2 <= desired_atpg_detections <= 10**

The *simulation_drop_limit* is set to 10.

- **guaranteed_atpg_detections > 10 OR desired_atpg_detections > 10**

The *simulation_drop_limit* is set to the greater of the *guaranteed_atpg_detections* or *desired_atpg_detections* values.

- **-REPort_incremental_statistics {ON | OFF}**

An optional switch and literal pair that enables the tool to report incremental multiple-detect test coverage statistics at the end of each multiple-detect ATPG loop. The default is for the tool to not report such statistics.

- **OFF**

An optional literal that disables the multiple detection of faults.

Examples

Example 1

The following example generates a 3-detection pattern set without EMD, where a fault is dropped from the simulation after 5 detections:

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1  
-simulation_drop_limit 5
```

Example 2

The following example generates a pattern set where each testable fault must be detected twice and should be detected at least 8 times:

```
set_multiple_detection -guaranteed_atpg_detections 2 -desired_atpg_detections 8
```

The simulation drop limit is automatically set to 10.

Example 3

The following example disables the multiple detection of faults:

```
set_multiple_detection off
```

Related Topics

[create_patterns](#)

[read_faults](#)

[report_environment](#)

[report_faults](#)

[report_statistics](#)

[set_fault_type](#)

[write_faults](#)

set_multiprocessing_options

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.

Usage

```
set_multiprocessing_options [-generic_delete string] [-generic_scheduler string]
[-license_timeout {integer | unlimited | no_queue}] [-lsf_heuristics {on | off}]
[-lsf_learning {on | off}] [-lsf_options string] [-multithreading {on | off}]
[-processors_per_grid_request integer] [-remote_shell {rsh | ssh}]
[-result_time_limit integer] [-scheduler_timeout {integer | unlimited}]
[-sge_options string]
```

Description

Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.

Note

 Take care to exclude any unsupported platforms that exist in your grid environment by specifying appropriate grid job submission constraints. For a list of supported platforms, refer to “[Supported Hardware and Operating Systems](#)” in *Managing Mentor Graphics Tessent Software*.

For more information about setting up for multiprocessing, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual* and “[The Tessent Tcl Interface](#)” in the *Tessent Shell User’s Manual*.

Arguments

- **-generic_delete *string***

An optional switch and string that specify a command to delete a slave process. You create a slave process via the generic job scheduler when you issue commands such as “[add_processors generic](#)”. The string consists of a command that the tool uses to delete a job with the generic job scheduler. It also contains a placeholder which has the format %*x*, where “*x*” is a single-word string. The tool looks for the single-word string in the output of the command specified for the generic jobs submission via the -generic_scheduler switch and expects to find a number following the string that represents the job number used by the generic scheduling system. The command specified by the -generic_scheduler switch is issued when each slave is added, and the number parsed by the tool is then stored and substituted for the placeholder in the generic_delete commands issued to the generic scheduling system to delete each slave.

If you specify, for example:

```
set_multiprocessing_options -generic_delete \
{ $SGE_ROOT/bin/lx24-x86/qdel %job }
```

Then running the command specified by the `-generic_schedule` switch should produce output that contains a job number following the word “job”, such as:

```
Your job 32335 ("mentortask") has been submitted
```

The tool parses the number after ‘job’ in the output and substitutes it for the placeholder `%job`, when it deletes the slave process. In this case, the tool will form the command:

```
$SGE_ROOT/bin/lx24-x86/qdel 32335
```

The placeholder can be any word, as long as it matches the word that is output prior to the job number in the output of the command specified by the `-generic_scheduler` switch. If that word is “task” instead of “job” as in the following:

```
Your task 33435 ("mentortask") has been submitted
```

Then you would specify a `generic_delete` command containing `%task` as in the following:

```
set_processing_options -generic_delete \
{ $SGE_ROOT/bin/lx24-x86/qdel %task }
```

The tool parses the number after ‘task’ in the output and substitutes it for the placeholder `%task`, when it deletes the slave process. In this case, the tool will form the command:

```
$SGE_ROOT/bin/lx24-x86/qdel 33435
```

Please note that the string that the tool looks for in the output is case sensitive. For example, if the scheduler command output was:

```
Task 3343 ("mentortask") has been submitted
```

The command specified by the `-generic_delete` switch would have to contain the string “%Task” rather than “%task”.

The tool uses this command to delete jobs when they are no longer needed, such as when you issue a `delete_processors` command, when the tool shuts down, or when the scheduler exceeds its timeout setting or is interrupted by Ctrl-C. This explicitly notifies the generic scheduling system that the job can be terminated or has already completed. If you use the generic scheduler without specifying a `generic_delete` command, the tool will simply tell the running slave processes to terminate and leave it up to the generic scheduling system to detect the termination of the slave processes.

- **-generic_scheduler *string***

An optional switch and string that specifies a command script that requests a slave process from the generic job scheduler (that is used for grid systems other than LSF or SGE). The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent `add_processors` commands, not currently

running processes. An incorrect specification can prevent the correct operation of the add_processors command.

- **-license_timeout {integer | unlimited | no_queue }**

An optional switch and non-negative integer that specifies the number of minutes the tool waits when attempting to acquire a license. The default value is 5 minutes.

integer—A non-negative integer that specifies the queuing time in minutes. A zero value will cause the tool to not wait for a license, the same behavior as specifying “no_queue”. The tool reports when it is unable to acquire a license in the specified amount of time.

unlimited—An optional string that specifies no time limit. The tool waits until a license becomes available.

no_queue—An optional string that specifies that the tool not wait for a license and reports when a license is not available immediately.

- **-lsf_heuristics {on | off}**

An optional switch and keyword that enables or disables LSF heuristics learning when you add a processor by LSF grid request. The tool uses LSF heuristics learning in an attempt to automatically choose machines on the LSF grid that are compatible with the tool if you did not specify an LSF “type” constraint with -lsf_options and if either LSF scheduler learning has failed or you disabled that feature using “-lsf_learning off.”

LSF heuristics learning may be able to determine tool compatibility based on the names of the types and models available and successfully identify some type/model combinations that the tool can use; however, this may also result in specifying incompatible combinations or miss some compatible combinations. If LSF heuristics learning is turned off or fails, then the tool submits the job anyway without specifying any type/model restrictions.

LSF heuristics learning is enabled by default. For more information about using this switch, refer to “[Processor Addition to the LSF Grid](#)” in the in the *Tessent Scan and ATPG User’s Manual*.

- **-lsf_learning {on | off}**

An optional switch and keyword that enables or disables LSF scheduler learning when you add a processor by LSF grid request. When this switch is on and you have not specified an LSF “type” constraint with -lsf_options, the tool asks the LSF system what machines and type/model combinations are available, after which the tool runs the **lsrun** command to log onto likely machines to determine which are compatible with the tool.

LSF scheduler learning cannot succeed if your LSF system is configured to prevent use of **lsrun**. If LSF scheduler learning is turned off or fails, the tool then uses LSF heuristics learning unless you have disabled this feature with “-lsf_heuristics off.”

LSF scheduler learning is enabled by default. For more information about using this switch, refer to “[Processor Addition to the LSF Grid](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-lsf_options** *string*

An optional switch and string that specifies command options for the tool to append to the job submission command for the LSF job scheduler. The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent add_processors commands, not to currently running processes. An incorrect specification can prevent the correct operation of the add_processors command.

When you add the “verbatim” key symbol at the beginning of the string, the tool interprets the rest of the line as the submission specification for each grid request it makes, overriding any memory and swap availability requirements it would otherwise attempt to add (see the note for the add_processors **hostname** argument). When you use this keyword, be sure to include any needed memory and swap requirements in the rest of the -lsf_options string. For more information, refer to “[Example 8](#)” on page 2274.

- **-multithreading** {on | off}

An optional switch and keyword that disables multithreading functionality. Multithreading is enabled by default.

- **-processors_per_grid_request** *integer*

An optional switch and integer value that specifies the maximum number of processors (slots) to group into one single request for the grid system, either for SGE, LSF, or GENERIC requests. The default value is “unspecified” (-1) because there are different default values for SGE, LSF, and GENERIC. The default for LSF requests is 4. The tool can use LSF job submission switches to ensure that this number of processors is requested per grid request. For SGE and GENERIC processor requests, the syntax for requesting a specific number of processor slots per grid request is site-specific, so the tool defaults to one processor per request.

If you want to specify the number of processors per request when using SGE or a generic grid scheduler, you must supply the needed syntax using the -sge_options, -generic_scheduler, and (optionally) the -generic_delete switches. You can do this by hardcoding a value in the -sge_options or -generic_scheduler string, or by using the pseudo variable %processors_per_grid_request, which allows you to change the value of processors_per_grid_request later without also having to change the corresponding grid command variable value. For an example of using this variable for the -lsf_options switch, refer to “[Example 3](#)” on page 2272.

The requested number of processors (for example, add_processors sge:20) results in the following number of grid requests: the requested number of processors divided by processors_per_grid_request. The result of the division is rounded up, and the last grid request can request fewer processors than the number of “processors_per_grid_request.”

Note



The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. For more information, refer to “[Multiprocessing to Reduce Runtime](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-remote_shell {rsh | ssh}**

An optional switch and keyword that specifies the shell command that the master host should use to create processes on slave hosts. Applies only when you specify hosts manually rather than using a job scheduler. By default, the tool uses the **rsh** command.

- **-result_time_limit *integer***

An optional switch that specifies a time limit in minutes for detecting non-responsive slaves while waiting for certain types of job results. If the tool detects that the time limit has been exceeded for a specific result needed to make further progress, the tool issues a warning about the non-responsive slave and waits again for the same result. (This gives you the opportunity to investigate and correct the problem.) If the **result_time_limit** is again exceeded, the tool kills the slave and reassigns the job to another processor, and the cycle starts over. In most cases, you do not have to change the default value. Note that if you set the value too low, the tool could possibly kill all of its slaves needlessly. A value of 0 disables the time limit, but otherwise the command does not accept a value of less than 15. The default value is 45.

- **-scheduler_timeout {*integer* | unlimited}**

An optional switch and non-negative integer value or the string “unlimited” that specifies a time limit allowed, in minutes, for the tool to attempt to add processors using a job scheduler. If the tool does not receive all the requested processors within the specified time, it will proceed with the processors it has received.

integer—A non-negative integer that specifies the time limit, in minutes, for the tool to attempt to add processors using a job scheduler. A zero value indicates the same as specifying the “unlimited” string. The default value is 10 minutes.

unlimited—An optional string that specifies that the tool waits until all requested processors are received from the grid.

- **-sgc_options *string***

An optional switch and string that specifies command options for the tool to append to the job submission command for the SGE or OGE job scheduler. The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent **add_processors** commands, not currently running processes. An incorrect specification can prevent the correct operation of the **add_processors** command.

Command Variables

The following variables are available for use in this command. You can define these variables in your dofile for convenience.

%command—Represents a substring you use with the **-generic_scheduler** switch to specify the location to substitute the Mentor Graphics command script that launches a slave process. For more information, refer to “[Example 5](#)” on page 2273.

%job—A placeholder, rather than a variable. For more information, on this placeholder, refer to the description of [-generic_delete *string*](#).

`%processors_per_grid_request` — Specifies the maximum number of processors (slots) to group into one single request for the grid system, either for SGE, LSF, or GENERIC requests. For more information, refer to “[Example 3](#)” on page 2272.

Tcl Notes

When using Tcl constructs with the `set_multiprocessing_options` command, the tool’s logfile reports the evaluated command and strips any Tcl special characters (quotes, braces, and brackets) from the logfile. If you need to replay a session from the logfile, you must add the Tcl syntax.

Enclosing any expression in braces (`{ }`) prevents the tool from evaluating the expression. Enclosing any expression in double quotes (`“ ”`) causes the tool to resolve any variables in the expression.

Examples

Example 1

The following example controls type and memory constraints that would normally be supplied automatically:

```
set_multiprocessing_options -lsf_options { -R \
    select[type==LINUX64 && mem> 200 && swp > 200 ] }
get_multiprocessing_option -lsf_options
-R select[type==LINUX64 && mem> 200 && swp > 200]
```

This setting would constrain LSF to select hosts meeting the specified requirements. The “type” specification also inhibits the automatic learning that would normally take place to discover all the type/model names in an LSF cluster. (This is useful if your site prevents the use of the `lrun` command or if the learning hits non-responsive hosts.) You can specify any legal LSF constraints for the installation, which overrides any related specifications the tool would have provided otherwise. It is up to you to supply syntactically and semantically correct LSF options.

Example 2

The following example specifies that LSF use queue *myqueue*:

```
set_multiprocessing_options -lsf_options "-q myqueue"
```

Example 3

The following example specifies that LSF reserve `%processors_per_grid_request` processors or slots:

```
set_multiprocessing_options -lsf_options "-n %processors_per_grid_request"
```

Note, however, that the `-n` specification would probably be only a subset of the `bsub` options assigned to the `-lsf_options` variable, and that the above example command line serves only as an example of how to use the `%processors_per_grid_request` variable. If you don’t use the `-n` switch in the `-lsf_options` string, the tool still specifies `-n` with the `processors_per_grid_request` value to the LSF job scheduler by default (value of 4). Note than when requesting additional processors per grid request, the number of grid requests decreases accordingly.

Example 4

The following example specifies that SGE impose a 6 hour time limit on slave processes:

```
set_multiprocessing_options -sgc_options {-l h_rt=6:00:00}
```

Example 5

The following example specifies an alternative interface to SGE for launching slave processes:

```
set_multiprocessing_options -generic_scheduler \
{ $SGE_ROOT/bin/lx*-amd64/qsub %command }
```

The *%command* represents a substring you must use with the *-generic_scheduler* switch to specify the location to substitute the Mentor Graphics command script that launches a slave process. Variables can be defined in the dofile to make this more convenient, as in the following example dofile excerpt:

```
set SUBMIT { $SGE_ROOT/bin/lx*-amd64/qsub }
set _multiprocessing_options -generic_scheduler "${SUBMIT} %command"
```

Example 6

The following example specifies an alternative interface to SGE for terminating slave processes:

```
add_processors generic
set_multiprocessing_options -generic_delete { $SGE_ROOT/bin/lx24-x86/qdel %job }
```

The *%job* is a placeholder that specifies a word, in this case “job”, that the tool uses to locate the job number in the generic scheduler’s output. The job number is substituted for the placeholder and used with the command *\$SGE_ROOT/bin/lx24-x86/qdel* when deleting processors.

The place holder does not need to be *%job*. It is whatever was used when the slave process was started. This is a variation of this example using a different placeholder, in this case, *%task*:

```
add_processors generic
set_multiprocessing_options -generic_delete { $SGE_ROOT/bin/lx24-x86/qdel %task }
```

Example 7

The following example specifies that the master host should use the **ssh** shell command to create processes on slave hosts (manual specification of hosts only):

```
set_multiprocessing_options -remote_shell ssh
```

Tip If you encounter errors trying to use SSH, refer to “[SSH Environment and Passphrase Errors](#)” in the *Tessent Scan and ATPG User’s Manual*.

Example 8

The following example specifies that LSF job requests should specify the use of hosts with type LINUX64 and should not specify any additional job constraints:

```
set_multiprocessing_options -lsl_options { verbatim -R "select[ type==LINUX64 ]" }
```

Related Topics[add_processors](#)[delete_processors](#)[get_multiprocessing_option](#)[report_multiprocessing_options](#)[report_processors](#)

set_net_dominance

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Specifies the fault effect of bus contention on tri-state nets.

Usage

`set_net_dominance Wire | And | Or`

Description

Specifies the fault effect of bus contention on tri-state nets.

The `set_net_dominance` command specifies the fault effect of bus contention on tri-state nets. This provides the capability to detect some faults on tri-state driver enable lines when those drivers connect to a tri-state bus. These faults would be ATPG-untestable unless the tool can use the Z-state for detection.

When using Tessent FastScan or Tessent TestKompress, the Wire behavior is the same where any different binary value results in an X-state.

The truth tables for each type of bus contention fault effect are given in Tables [6-13](#), [6-14](#), and [6-15](#). This command does not affect Good machine behavior.

Table 6-13. WIRE Bus Contention Truth Table

	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Table 6-14. AND Bus Contention Truth Table

	X	0	1	Z
X	X	0	X	X
0	0	0	0	0
1	X	0	1	1
Z	X	0	1	Z

Table 6-15. OR Bus Contention Truth Table

	X	0	1	Z
X	X	X	1	X
0	X	0	1	0
1	1	1	1	1
Z	X	0	1	Z

Arguments

- **Wire**

A literal that specifies for the tool to use unknown behavior for the fault effect of bus contention on tri-state nets. This is the default behavior upon invocation of the tool.

- **And**

A literal that specifies for the tool to use wired-AND behavior for the fault effect of bus contention on tri-state nets.

- **Or**

A literal that specifies for the tool to use wired-OR behavior for the fault effect of bus contention on tri-state nets.

Examples

The following Tessent FastScan example specifies that the fault effect on tri-state nets is wired-AND during ATPG:

```
set_net_dominance and  
set_system_mode analysis  
create_patterns
```

set_net_resolution

Context: all contexts

Mode: setup

Specifies the behavior of multi-driver nets.

Usage

`set_net_resolution Wire | And | Or`

Description

Specifies the behavior of multi-driver nets.

The `set_net_resolution` command specifies the behavior of non-tri-state, multi-driver nets. The default upon invocation of the tool is `Wire`, which requires all inputs be at the same value to achieve a value. If you can model your nets using the `And` or `Or` option, you can improve your test coverage results.

Arguments

- **Wire**

A literal that specifies for the tool to use unknown behavior for non-tri-state, multi-driver nets. This requires all inputs to be at the same value to achieve a value other than X. This is the default upon invocation of the tool.

If you are using Tessent FastScan or Tessent TestKompress with E9 rule checking enabled, the tools do not perform checking on wire gates if you use the `set_net_resolution` command to change their behavior to AND or OR.

- **And**

A literal that specifies for the tool to use wired-AND behavior.

- **Or**

A literal that specifies for the tool to use wired-OR behavior.

Examples

The following Tessent FastScan example specifies that the behavior of non-tri-state, multi-driver nets is wired-AND during ATPG:

```
set_net_resolution and  
set_system_mode analysis  
create_patterns
```

set_number_shifts

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Sets the number of shifts for loading or unloading the scan chains.

Usage

set_number_shifts *shift_number*

Description

Sets the number of shifts for loading or unloading the scan chains.

Note

 For diagnosis, you can only use the set_number_shifts command to modify the number of shifts for an EDT design. There is no support in the diagnosis tool for non-EDT designs.

By default, the number of shifts used for loading or unloading the scan chains in a scan group is the same as the largest number of scan cells in any scan chain in that scan group. The tool determines this number once it traces all scan chains. However, you can use the set_number_shifts command to increase the default number.

You normally issue this command in setup mode. If the specified number of cycles is more than the number determined during DRC, the tool adds the extra cycles as overshift cycles. If the specified number of cycles is less than required, the tool issues an error.

If EDT is enabled, you can issue this command in either setup or analysis mode. The number of shift cycles specified in analysis mode does not impact the pattern generation process, which uses the number determined during DRC. The generated external patterns are padded with the additional cycles when they are saved using the “[write_patterns -edt_external](#)” command.

For edt_internal patterns, you must use the parameter file keyword ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS in order to apply the number of shift cycles specified by set_number_shifts. For more information, refer to “[ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS](#)” on page 3945.

Here are some considerations for using the set_number_shifts command:

- The set_number_shifts command is mutually exclusive with the “set_edt_options -OVERShift_cycles *integer*” command. If you attempt to issue both commands in the same session, the tool issues an error and resets the number of shifts to zero.
- The set_number_shifts command does not affect parallel test benches.
- The set_number_shifts command has no effect on WGL patterns if both the WGL_FULL_CHAIN and WGL_FULL_SCANGROUP parameter file keywords are

set to 0. By default, WGL_FULL_CHAIN is set to 1 and WGL_FULL_SCANGROUP is set to 0.

Arguments

- ***shift_number***

Specifies the number of shifts. If the number specified is smaller than the default number determined by the tool, the tool issues an error message.

Examples

Example 1

The following example sets the number of shifts for loading or unloading the scan chains to 2.

set_number_shifts 2

Example 2

The following example adjusts the number of shifts for a given pattern set and runs diagnosis:

```
set_number_shifts 100
read_patterns my_patterns/pat.wgl
diagnose_failures flogs/fail_1.log
```

Related Topics

[set_edt_options](#)

set_observation_point

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the observation point for random pattern fault simulation.

Usage

`set_observation_point Master | SLave | SHadow | Clockpo`

Description

Specifies the observation point for random pattern fault simulation.

The `set_observation_point` command specifies whether Tessent FastScan observes master latches, slave latches, shadow latches, or clock primary outputs during random pattern fault simulation. If you select Master, Slave, or Shadow, Tessent FastScan also observes the primary outputs that do not connect to clock lines. The default behavior upon invocation of Tessent FastScan is Master.

Arguments

- **Master**

A literal that specifies observation of master latches and normal primary outputs. This is the default behavior upon invocation of Tessent FastScan.

- **SLave**

A literal that specifies observation of slave latches and normal primary outputs.

- **SHadow**

A literal that specifies observation of observable shadow latches and normal primary outputs.

- **Clockpo**

A literal that specifies observation of only primary outputs directly connected to clocks.

Examples

The following Tessent FastScan example specifies slave latches as the observation point for random pattern fault simulation:

```
set_system_mode analysis
set_observation_point slave
add_faults -all
simulate_patterns -source random
```

Related Topics

[set_capture_clock](#)

[simulate_patterns](#)

set_random_patterns

set_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Ignores any fault effects that propagate to the primary output pins you select.

Usage

```
set_output_masks Off | ON [-Bidi_exclude] [-Exclude pin.pathname...]
```

Description

Ignores any fault effects that propagate to the primary output pins you select.

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command allows you the ability to flag all or a portion of the primary output pins that do not have strobe capability (are not observable). The tool classifies the faults whose effects only propagate to that observation point as Atpg_Untestable (AU).

You can use set_output_masks in setup mode or analysis mode, but not both. The tool behaves as follows:

- When output masks are enabled in setup mode and you disable them in analysis mode, the tool reports, “Error: Output masks enabled in setup mode cannot be disabled in analysis mode.”
- When output masks are enabled in setup mode and you enable them in analysis mode, the tool reports, “Note: Output masking was already enabled in setup mode.”
- When output masks are enabled in analysis mode and you return to setup mode, the tool disables the output masks.

You can use the [report_output_masks](#) command to display the current setting of the output masks.

Arguments

- **Off**

A required literal that specifies for the tool to use primary output pins as the observe points during the fault detection process. In addition, the tool will give possible credit for fault effects that reach a pin that the pattern set specifies as being at an unknown value. This is the default behavior upon invocation.

- **ON**

A required literal that specifies for the tool to mask the simulated primary output pin values with Xs for all patterns. As a result, the tool does not give any credit for fault effects that reach those masked primary output.

- **-Bidi_exclude**

An optional switch that modifies how the command treats bidirectional pins.

Note

 The meaning of the **-Bidi_exclude** switch is different for each setting (“off” or “on”) of the command. It basically specifies to do the opposite, for bidirectional pins, of what is done for other pins.

When used with the *Off* literal, the **-Bidi_exclude** switch specifies for the tool to `set_output_masks "on"` for bidirectional pins. Thus, the command “`set_output_masks off -bidi_exclude`” results in the tool masking bidirectional pins and no others.

Conversely, if you use the *On* literal, then adding the **-Bidi_exclude** switch causes the tool to `set_output_masks "off"` for bidirectional pins. The command “`set_output_masks on -bidi_exclude`”, therefore, results in the tool masking all pins except bidirectional pins.

- **-Exclude pin_pathname...**

An optional switch and repeatable string that modifies how the command treats a specific pin.

Note

 Similar to the **-Bidi_exclude** switch, the **-Exclude** switch specifies for the tool to do the opposite, for each specified pin, of what is done for other pins.

When used with the *Off* literal, the **-Exclude** switch will cause the tool to `set_output_masks "on"`, but only for the specified pin(s). The command “`set_output_masks off -exclude my_pin`”, for example, will result in the tool masking the pin whose pathname is “`my_pin`” and not masking any others.

Conversely, when used with the *On* literal, the **-Exclude** switch will cause the tool to `set_output_masks "off"` for the specified pin(s). In this case, “`set_output_masks on -exclude my_pin`” will apply masking to all pins except the pin “`my_pin`”.

Examples

The following example reports all the primary outputs of a design having one primary output, / `y_inout`, that is bidirectional. It then specifies to mask all the primary outputs *except* any that are bidirectional, and the three pins, `y_out[3]`, `y_out[2]`, and `x_out[1]`. The example then displays the masked primary output pins. As you would expect, none of the pins excluded from masking are listed.

report_primary_outputs

```
SYSTEM: /x_out [3]
SYSTEM: /x_out [2]
SYSTEM: /x_out [1]
SYSTEM: /x_out [0]
SYSTEM: /y_out [3]
SYSTEM: /y_out [2]
SYSTEM: /y_out [1]
SYSTEM: /y_out [0]
SYSTEM: /my_out
SYSTEM: /y_inout
```

set_output_masks on -bidi_exclude -exclude y_out[3] y_out[2] x_out[1]

report_output_masks

```
TIEX /x_out [3]
TIEX /x_out [2]
TIEX /x_out [0]
TIEX /y_out [1]
TIEX /y_out [0]
TIEX /my_out
```

Related Topics

[add_output_masks](#)

[delete_output_masks](#)

[report_environment](#)

[report_output_masks](#)

set_parallel_load_subchains

Context: dft -edt, patterns -scan, patterns -scan_retargeting,
patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the internal parallel access functionality is enabled or disabled.

Usage

```
set_parallel_load_subchains [OFF|ON]
```

Description

Specifies whether the internal parallel access functionality is enabled or disabled.

By default, internal parallel access is disabled. In this case, when a netlist contains a subchain in a library model, the parallel test bench can only apply serial shifts to load the scan data and compare the capture values because internal pin names (where the parallel value is forced and observed) of the library models cannot be accessed. This limitation can increase parallel simulation runtime and significantly increase the size of the parallel test bench file.

If this command is enabled in Setup mode, the parallel test bench is written to access internal pin names of the library models. When you exit Setup mode, a DRC warning is issued if the netlist contains a subchain. In this case, you should enable this command. You can use the + command to display detailed information about the subchains in the netlist.

Arguments

- OFF
An optional literal that disables internal parallel access. This is the default.
- ON
An optional literal that enables internal parallel access. This command can be disabled or re-enabled in analysis mode to write out different parallel test benches if necessary, but only if it was initially enabled in Setup mode. If this command was not enabled in Setup mode, you cannot change the setting in analysis mode because internal access points can only be learned during DRC when switching from Setup mode to a non-Setup mode.

Note

 When this command is enabled, extra memory is required for storing the internal names of the library cells. Also, internal pin names of the Tessent Cell library models must match the internal pin names of the Verilog library model; otherwise, test bench compile errors or incorrect results from a Verilog simulation can occur. If you used LibComp to translate the Verilog libraries to ATPG libraries, then internal pin names should match because LibComp preserves these internal names.

Examples

The following example enables internal parallel access in Setup mode, sets the system mode to ATPG, creates test patterns, and saves the test patterns with parallel loading of subchains.

```
set_parallel_load_subchains on
set_system_mode analysis
create_patterns
write_patterns trans_pat.gz -verilog -parallel
```

If the test bench is not compiled because the internal pin names of the Verilog library do not match those of the Tessent Cell library, parallel loading can be disabled. Then the test bench loads the subchain cells serially.

```
set_parallel_load_subchains off
write_patterns trans_pat2.gz -verilog -parallel
```

Related Topics

[report_environment](#)

[report_scan_chains](#)

set_pathdelay_holdpi

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.

Usage

set_pathdelay_holdpi OFF | ON

Description

Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.

The set_pathdelay_holdpi command lets you specify for the tools to ignore non-clock primary input changes after the first force in each pattern.

Arguments

- **OFF**
A literal specifying that the tool can change non-clock primary input values at any time. This is the default behavior upon invocation of the tool.
- **ON**
A literal that specifies for the tool not to change non-clock primary input values after the launch of the transition into the path.

Examples

The following example enables Tessent FastScan to ignore changes to non-clock primary inputs:

set_pathdelay_holdpi on

Related Topics

[set_transition_holdpi](#)

set_pattern_buffer

Context: unspecified, dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Stores run-time pattern data in temporary files in the directory specified.

Usage

set_pattern_buffer {*directory_name...*} | -Off

Description

Stores run-time pattern data in temporary files in the directory specified.

The **set_pattern_buffer** command allows you to specify a directory where the tool can store temporary run-time pattern data. When the tool accesses a pattern, it is then automatically retrieved from the specified directory, as opposed to virtual memory, conserving the memory for other tool processing tasks. This provides a run-time processing advantage when there is not ample swap space.

You can specify multiple directories. As individual buffer files approach their file size limit or volume space runs out, the tool searches the specified directory list for available space. The tool deletes these buffer files when you exit the application.

Note

 Buffer files may be multiple gigabytes large. Choose directories on volumes that have enough disk space to handle expected amount of pattern data.

Arguments

- ***directory_name***

A required repeatable string that specifies the full path directory names in which the tool saves buffer files, thus enabling the usage of temporary buffer files for pattern data. Reissuing this command with additional directories causes the tool to add the directories to the current list of directories available for buffer files.

- **-Off**

A required switch that disables the use of temporary buffer files, causing the tool to save run-time data in virtual memory. This is the default upon invocation. Disabling the usage of pattern buffers can only be done when there is no pattern data, such as after issuing the **reset_state** or **set_system_mode** commands.

Examples

The following example places buffer files in the specified directories.

```
set_pattern_buffer /usr/jdoe/buffers /usr2/jdoe/morebuffers
```

Related Topics

[report_resources](#)

set_pattern_classification

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the order in which to store test patterns.

Usage

`set_pattern_classification ON | OFF`

Description

Specifies the order in which to store test patterns.

When set to “on,” each pattern is stored by its classification, according to the following ordered list:

basic
read only (a type of basic pattern)
ram sequential
clock sequential and multi-load

When set to “off,” test patterns are stored in the order in which they were generated. Note that when simulating the external patterns (by issuing the [simulate_patterns](#) command), the tool automatically turns off pattern classification.

Arguments

- **ON**
A literal that enables pattern classification. If the current internal pattern set is not classified, the pattern is automatically classified after the command is executed.
- **OFF**
A literal that turns off pattern classification. When pattern classification is disabled, patterns are stored in the same order in which they were generated.

Examples

The following example stores patterns in the order in which the tool generates them:

`set_pattern_classification off`

Related Topics

[write_patterns](#)

[set_pattern_filtering](#)

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Creates a temporary set of sampled scan patterns.

Usage

```
set_pattern_filtering [-EXternal] -Off
| [-LList pattern_index...]
| [{-PATtern_type keyword}...]
| [{-NOPattern_type keyword}...]
[-SAMple_per_type integer]
[-CAPture]
[{-RANge begin_int end_int}...]
[{-NORange begin_int end_int}...]
[{-NAmed_capture_procedures {procedure_name ... | ALL}}}
| {[{-CLocks clock_name...} [-NO_CLocks clock_name ...]}]
| [-RESTRICT_CLocks clock_name...}]
[{-CAPTURE_Power operator percentage [ST_Peak | ST_Average | WSA_Peak
| WSA_Average] [-INSTance object_name...] [-MODULE module_name...]...]}
[{-SHIFT_Power {LOAD | RESPonse} operator percentage] [-INSTance object_name...]
[-MODULE module_name...]...]
```

Description

Creates a temporary set of sampled scan patterns.

You can then save the sampled patterns in all supported formats with the [write_patterns](#) command. To restore the original non-sampled patterns, issue `set_pattern_filtering -Off`. If the content of the original pattern set changes while pattern filtering is on, pattern filtering is turned off automatically.

By default, this command preserves the original pattern order. When you use the -list switch, the user-specified pattern order will be enforced. Also, the filtering and ordering only applies to scan (ATPG) patterns, and is not performed for chain test patterns.

Note

 Because MacroTest patterns are order dependent, they are sampled sequentially, beginning at the first pattern. Non-MacroTest patterns are re-ordered.

Arguments

- **-EXternal**

An optional switch that applies sampling to external patterns. Before using this switch, you must specify the name of the external pattern file with the [read_patterns](#) command. The tool also re-orders the patterns.

- **-Off**

An optional switch that restores the original non-sampled patterns.

- **-L***ist pattern_index...*

An optional switch and repeatable, non-negative, integer pattern indexes that creates a sample consisting of patterns with the specified pattern indexes, and orders the sampled patterns in the same order the indexes appear (left to right) in the command. The *pattern_index* refers to original pattern indexes in the pattern set before filtering by the `set_pattern_filtering` command. This switch cannot be combined with any other pattern filtering options.

This switch enables you to arrange patterns in any order. For example, during IDDQ test, if the current of every pattern is measured to sort them in order of increasing current, the result is a list of pattern indices (for example, 13, 2, 24, and so on) that does not match any of the tool's built-in ordering criteria. You could use this switch to order the patterns according to the IDDQ sort.

- **-PATtern_type keyword**

An optional switch and keyword pair that applies sampling based on the keyword pattern type. Multiple “-Pattern_type keyword” pairs are allowed; however, you cannot combine this switch with the -Nopattern_type switch in the same command. Keywords are listed in [Table 6-16](#). If you specify neither this switch nor the -Nopattern_type switch, the default is to sample all pattern types.

Valid pattern type keywords are:

Table 6-16. Keywords for Pattern Types

Pattern Type	Keyword
Basic	BASIC
Clock PO	CLOCK_Po
Clock Sequential	CLOCK_Sequential
MacroTest	MacroTest
Multi-load	MULTI_load
RAM Sequential	RAM_sequential

- **-NOPattern_type keyword**

An optional switch and keyword pair that specifies to not sample the keyword pattern type. Multiple “-Nopattern_type keyword” pairs are allowed; however, you cannot combine this switch with the -Pattern_type switch in the same command. Keywords are listed in [Table 6-16](#).

- **-SAMple_per_type integer**

An optional switch and positive integer pair that specifies how many patterns of each type to sample. The default is 1.

- **-CAPture**

An optional switch that samples the patterns by capture clock rather than pattern type. It samples patterns based upon the capture clock of each capture cycle so that each clock pulse in different capture cycles are sampled n times, where n is selected by the “-sample_per_type n” option (default is 1). See [Example 9](#).

If you specify the -Capture switch with the -Pattern_type switch, the command only samples patterns by capture clock within the specified pattern types.

- **-RANge beg_int end_int**

An optional, repeatable switch and pair of integers that creates a sample of patterns beginning with pattern beg_int and ending with pattern end_int. Patterns are numbered beginning with 0; for example, to restrict sampling to the first 100 patterns, use “-range 0 99.”

- **-NORange beg_int end_int**

An optional, repeatable switch and pair of integers that specifies to not apply sampling to the range of patterns beginning with pattern beg_int and ending with pattern end_int. Patterns are numbered beginning with 0; for example, to exclude patterns 51 through 60 from sampling, use “-norange 51 60.” You can use this switch to exclude sub-ranges within sampled ranges specified with the -Range switch.

- **-NAMED_capture_procedures procedure_name... / ALL**

An optional switch and repeatable string or literal pair that samples patterns for either the specified named capture procedures or all named capture procedures.

The number of patterns sampled depends on the number of samples specified using the -sample switch. If the -sample switch is not used, all the patterns that use the specified named capture procedures are sampled.

- **-Clocks clock_name...**

An optional switch and repeatable string pair that samples only patterns that use one or more of the specified clocks. This includes patterns that use unspecified clocks if they also use at least one of the specified clocks. For example, “set_pattern_filtering -clocks clk1 clk2 clk3” will sample a pattern that pulses clk1, clk2, or clk3. Patterns that do not pulse any clock are not sampled.

You cannot combine this switch with the -Restrict_clocks switch in the same command.

- **-NO_Clocks** *clock_name...*

An optional switch and optional, repeatable, string pair that samples only patterns that do not use any of the specified clocks. To obtain a sample that includes all the patterns that do not pulse any clock, omit the *clock_name* argument.

Note

 If you issue “`set_pattern_filtering -no_clocks clk1 clk2`,” where clk1 and clk2 are the only clocks pulsed by any pattern, the filtering result is the same as omitting the clock names and simply issuing “`set_pattern_filtering -no_clocks`.”

You cannot combine this switch with the **-Restrict_clocks** switch in the same command. If you omit the *clock_name* argument, you cannot combine this switch with the **-Clocks** switch in the same command.

- **-RESTRICT_Clocks** *clock_name...*

An optional switch and repeatable string pair that samples only patterns that use the specified clocks, or a subset of these clocks, exclusively. Patterns that use any unspecified clocks are not sampled, even if they also use some or all of the clocks specified. For example, “`set_pattern_filtering -restrict_clocks clk1 clk2 clk3`” will not sample a pattern that pulses clk1, clk2 and clk4. Patterns that do not pulse any clock are not sampled.

You cannot combine this switch with the **-Clocks** or **-No_clocks** switch in the same command.

- **-CAPTURE_Power**

Optional switch that filters test patterns based on switching activity during capture.

- **operator**

Optional literal that specifies an operation for test pattern filtering based on switching activity during shift or capture. Literal options include:

< — Less than

> — Greater than

<= — Less than or equal to

>= — Greater than or equal to

- **percentage**

Optional integer that specifies a switching threshold percentage that applies to filtering test patterns based on capture or shift power. The specified integer must be positive.

- **ST_Peak | ST_Average | WSA_Peak | WSA_Average**

Optional literal that filters the test patterns based on a specified metric. These options can only be used for switching activity during capture. Literals include:

ST_Peak — Peak of state transitions. Default.

ST_Average — Average of state transitions.

- WSA_Peak — Peak of weighted switching activity.
- WSA_Average — Average of weighted switching activity.
- **-INSTance *instance.pathname*...**
An optional switch and repeatable string that specifies one or more target instance(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The *instance.pathname* may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.
 - **-MODULE *module.name*...**
An optional switch and repeatable string that specifies one or more target module(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The *module.pathname* may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.
 - **-SHIFT_Power {LOAD | RESPonse}**
An optional switch and literal that filters test patterns based on switching activity during shift. Literal options include:
LOAD — filters based on switching during test pattern loading.
RESPonse — filters based on switching during the circuit response to test patterns.

Examples

Example 1

This example assumes ATPG has just been run, resulting in 125 patterns of three different types, including clock PO. There are two capture clocks. The example first saves a sample consisting of four clock PO patterns to a separate pattern file in TITDL format. The example then samples two patterns for each pattern type. Next, the same command is repeated with the addition of -Capture. This causes the tool to sample two patterns per type as before, and then add as necessary, additional patterns to ensure the sampled pattern set contains at least two samples per capture clock.

```
report_pattern_filtering
// Pattern filtering is off.

set_pattern_filtering -pattern_type clock_po -sample_per_type 4
// Pattern filtering is on: 1 pattern type(s),
// 4 selected pattern(s), 125 original pattern(s).

write_patterns clock_po.tdl -titdl -replace
// Note: pattern filtering is on, #-saved-pattern(s) = 4,
// #-total-pattern(s) = 125

set_pattern_filtering -sample_per_type 2
```

set_pattern_filtering

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
// 6 selected pattern(s), 125 original pattern(s).
```

set_pattern_filtering -capture -sample_per_type 2

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
// 7 selected pattern(s), 125 original pattern(s).
```

report_pattern_filtering

Current pattern filtering configuration

```
-----
original_patterns (internal) = 125
current_filtered_pattern(s) = 7
sampled_patterns_per_type = 2
sample_capture_clock = on
sample_pattern_type(s) = all
-----
```

Example 2

This example creates a sample of patterns 0 through 4, 6 through 10, and 20 through 29.

set_pattern_filtering -range 0 10 -norange 5 5 -range 20 29

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 20 selected pattern(s),
// 125 original pattern(s).
```

report_pattern_filtering

Current pattern filtering configuration

```
-----
original_patterns (internal) = 125
current_filtered_pattern(s) = 20
sampled_patterns_per_type = all
sample_capture_clock = off
sample_pattern_type(s) = all
include_range(s) = [20, 29], [0, 10]
exclude_range(s) = [5, 5]
-----
```

Example 3

This example creates a sample of patterns that use one or more of the clocks clk1, clk2, and clk3 and other clocks except for clk4; patterns that use clk4 are not sampled.

set_pattern_filtering -clocks clk1 clk2 clk3 -no_clocks clk4

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 6 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//      pre-
//      filter
// patt # patt #  type    cycles loads obs_proc cap_proc capture_clk_seq
// -----
//   0     0  basic      1      1      -      -  [clk3]
//   1     3  basic      1      1      -      -  [clk1]
//   2     5  clk_seq    2      1      -      -  [clk1]  [clk2]
//   3     6  clk_seq    2      1      -      -  [clk2]  [clk3]
//   4     9  clk_seq    2      1      -      -  [clk2]  [clk5]
//   5    18  clk_seq    2      1      -      -  [clk2]  [clk6]
```

Notice that when pattern filtering is in effect, the [report_patterns](#) command displays the index numbers of the patterns prior to filtering as well as after filtering.

Example 4

This example creates a sample of patterns that do not pulse any clock.

set_pattern_filtering -no_clocks

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 2 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//      pre-
//      filter
// patt # patt #  type    cycles loads obs_proc cap_proc capture_clk_seq
// -----
//   0     1  basic      1      1      -      -  [-]
//   1     2  basic      1      1      -      -  [-]
```

Example 5

This example creates a sample of patterns 54, 106, 34 and 125, in that order.

set_pattern_filtering -list 54 106 34 125

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 4 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//      pre-
//      filter
// patt # patt #  type    cycles loads obs_proc cap_proc capture_clk_seq
// -----
//   0     54  basic      1      1      -      -  [clk2,clk4]
//   1    106  clk_seq    3      1      -      -  [clk1]  [clk3]  [c
//   2     34  mult_ld    4      3      -      -  [clk1]
//   3    125  clk_seq    2      1      -      -  [clk1]  [clk2]
```

Example 6

This example creates a sample of patterns that use the specified named capture procedures TranPtrn01 and TranPtrn03.

```
set_pattern_filtering -named_capture_procedures TranPtrn01 TranPtrn03
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 17 selected pattern(s), 32 original
// pattern(s).

report_patterns

//      pre-
//      filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// -----
//    0    0  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    1    1  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    2    2  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    3    3  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    4    4  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    5    5  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    6    6  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    7    7  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    8    8  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//    9    9  clk_seq  2      1      -  TranPtrn01 [clk] [clk]
//   10   19  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]
//   11   20  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]
//   12   21  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]
//   13   22  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]
//   14   23  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]
//   16   25  clk_seq  2      1      -  TranPtrn03 [clk] [hclk]

report_pattern_filtering

Current pattern filtering configuration
-----
original_patterns (internal) = 32
current_filtered_pattern(s) = 17
sampled_patterns_per_type = all
sample_capture_clock = off
selected named capture proc = TranPtrn01, TranPtrn03
sample_pattern_type(s) = all
-----
```

Example 7

This example creates a sample of two patterns for each of the specified named capture procedures TranPtrn01 and TranPtrn03 and then saves the sampled patterns.

```
set_pattern_filtering -sample 2 -named TranPtrn01 TranPtrn03
write_patterns pat_TranPtrn01_TranPtrn03.ascii
```

Example 8

This example creates a sample of two patterns for all the named capture procedures and then saves the sampled patterns.

```
set_pattern_filtering -sample 2 -named ALL
write_patterns pat_all.ascii
```

Example 9

This example shows a pattern report before and after filtering the patterns using the -capture option.

report_patterns

```
//  
//   pattern #  type          cycles  load  observe_proc  capture_proc  capture_clock_sequence  
//   -----  -----  -----  -----  -----  -----  
//   0      basic           1       1      -            -            [-]  
//   1      basic           1       1      -            -            [-]  
//   2      basic           1       1      -            -            [-]  
//   3      basic           1       1      -            -            [-]  
//   4      basic           1       1      -            -            [sclk]  
//   5      basic           1       1      -            -            [sclk]  
//   6      basic           1       1      -            -            [sclk]  
//   7      basic           1       1      -            -            [sclk]  
//   8      basic           1       1      -            -            [sclk]  
//   9      basic           1       1      -            -            [reset_]  
//  10     basic           1       1      -            -            [sclk]  
//  11     basic           1       1      -            -            [sclk]  
//  12     basic           1       1      -            -            [sclk]  
//  13     basic           1       1      -            -            [sclk]  
//  14     basic           1       1      -            -            [sclk]  
//  15     basic           1       1      -            -            [reset_]  
//  16     basic           1       1      -            -            [sclk]  
//  17     basic           1       1      -            -            [sclk]  
//  18     clock_sequential 2       1      -            -            [sclk]  [clk12]  
//  19     clock_sequential 2       1      -            -            [sclk]  [clk12]  
//  20     clock_sequential 2       1      -            -            [reset_]  [clk12]
```

set_pattern_filtering -capture

```
// Internal pattern filtering is on: 4 selected pattern(s), 21 original
pattern(s).
```

report_patterns

```
//  
//   pre-filtering  
//   pattern #  pattern #  type  cycles  load  observe_proc  capture_proc  capture_clock_sequence  
//   -----  -----  -----  -----  -----  -----  -----  
//   0        0      basic   1       1      -            -            [-]  
//   1        4      basic   1       1      -            -            [sclk]  
//   2        9      basic   1       1      -            -            [reset_]  
//   3       18      clock_sequential 2       1      -            -            [sclk]  [clk12]
```

Notice that each clock pulsed in different capture cycles is sampled once, since the default sample per type is one.

Related Topics

[order_patterns](#)

[report_environment](#)

[report_pattern_filtering](#)

[report_patterns](#)

[write_patterns](#)

[read_patterns](#)

set_pattern_source

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Specifies the source of the test patterns.

Usage

When EDT is Off

```
set_pattern_source Internal | {Random | Bist | {{External filename}  
[-APPend | -Replace] [-Binary | -ASci | -STil | -WGl]}  
[-MASK mask_filename [-MASKFILE pattern_offset number]]  
[-PREserve_masked_observe_points] [-NOPadding] [-Store_patterns] [-ALL_patterns]}
```

When EDT is On

```
set_pattern_source Internal | {{External filename}  
[-APPend | -Replace] [-Binary | -ASci | -STil | -WGl]  
[-MASK mask_filename [-MASKFILE pattern_offset number]]  
[-PREserve_masked_observe_points] [-Store_patterns] [-ALL_patterns]}
```

Description

Specifies the source of the test patterns.

Note

 The set_pattern_source command followed by the “run” command has been mostly superseded by the [read_patterns](#) and [simulate_patterns](#) commands. For “set_pattern_source internal” and “run”, the replacement command is [create_patterns](#). Mentor Graphics recommends you use the newer commands.

Arguments

- **Internal**

A literal that specifies the internal set of patterns. This is the default upon invocation of the tool.

- **Random**

A literal that specifies the random patterns, capture clock, and observation point specified previously.

- **Bist**

A required literal that specifies BIST patterns.

There must be at least one defined LFSR or an error displays. The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup.

- **External *filename***

A literal and string pair that specifies the set of patterns contained in *filename* when performing a simulation run. For simulation, the specified patterns are loaded into an external pattern set database.

In analysis system mode, the tool adds effective patterns from a simulation run to the internal pattern set, if there is at least one undetected fault in the fault list. When you use the -All_patterns switch, the tool adds all simulated patterns from the external pattern set to the internal pattern set, not just the effective patterns.

In analysis mode, a simulation run does not alter the internal pattern set.

The external patterns can be in ASCII, binary, STIL, or WGL test pattern format.

Note



You may notice a drop in test coverage when using an external pattern set as compared to using generated patterns. This is an artificial drop.

There is an issue regarding artificial drops in test coverage due to redundant faults. When you run ATPG, the tool classifies some faults as redundant (RE). The tool knows these faults are untestable when calculating test coverage. If you restart the tool, or go back to Setup mode before reading the patterns back in, the faults that were previously classified as RE will now be classified as UC or UO. Now, the tool does not know these faults are untestable when calculating test coverage because faults cannot be proven redundant by fault simulation. The tool will only fault simulate the external patterns. It will not target undetected faults and try to prove them redundant as it does when the pattern source is internal. Therefore, you may see a slight drop in test coverage. This drop is an artificial drop and coverage still remains the same.

The external pattern formats are described in the [Tessent Scan and ATPG User's Manual](#).

- **-APPend**

An optional switch that specifies for the tool to append the patterns contained in filename to the patterns in the tool's current external pattern set.

- **-Replace**

An optional switch that specifies for the tool to replace its current external pattern set with the patterns contained in filename. This is the default.

- **-Binary**

An optional switch that specifies the external test pattern set is in binary format. This is used when reading in a file saved with the write_patterns -BInary command.

If neither -Binary or -Ascii options are specified, the tool tries to open and process the file as a binary file. If this is unsuccessful, the tool then tries to open and process the file as an ASCII file.

- [-ASci](#)

An optional switch that specifies the external test pattern set is in ASCII format. You cannot use this option with the Internal pattern source.

- [-WGl](#)

An optional switch specifying that the external test pattern set is in WGL format. You must use this option if you specify external patterns that are in WGL format.

Restrictions:

- For Tessent FastScan and Tessent TestKompress, you must load parallel scan patterns, not serial. Patterns produced by MacroTest cannot be read back in. For more information, see the note under the External argument.
- If you re-simulate scan patterns, only serial patterns are allowed.

- [-STil](#)

An optional switch specifying that the external test pattern set is in STIL format. You must use this option if you are loading STIL functional patterns into the tool.

Restriction:

- You must load parallel scan patterns, not serial.
- The patterns must be STIL patterns originally created in these tools.

Analysis of the STIL patterns conforms to the IEEE 1450.0 specification. The following features from the IEEE 1450.0 specification are not supported:

- Multiple data elements per test cycle (See section 5.7 of IEEE 1450.0)
- Non-cyclized test data (See section 5.9 of IEEE 1450.0)
- Goto statement referring to a previous pattern label (See section 22.8 of IEEE 1450.0)
- MatchLoop statement (See section 22.7 of IEEE 1450.0)

Analysis of the STIL patterns only supports waveform descriptions in the WaveformTable that Mentor Graphics tools currently support. Therefore, a waveform description should not have more than two event edges, unless it is describing a double pulse. All other waveforms that have more than two event edges are not allowed. Also, timing expressions are limited to using only the +, -, *, /, (, and) operators on constants or variables that have time-based units.

- [-MAsk](#)

An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns,

use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

The tool obtains the masking information from a text file you create (see the `mask_filename` argument).

This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using “`write_patterns -external`,” you will not see the masking information from the mask file reflected in the saved patterns; the saved patterns will be exactly the same as the original patterns.

To save a pattern set that incorporates the masking information, use the `run` command to perform a simulation of the external pattern set you specified with “`read_patterns -preserve_masked_observation_points`.” Then use the `write_patterns` command *without* the `-External` switch to write the resulting internal pattern set to a file.

Tip

 When you use the `-Mask` switch, the tool automatically executes the `-Store_patterns` switch with it; you do not need to include an explicit `-Store_patterns` in your command line.

- *mask_filename*

An optional string used with the `-Mask` switch to specify a mask file. The mask file is a text file you create that lists the patterns and locations you want masked. Use the statements and syntax described in [Table 6-17](#) to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

Note

 Tessent TestKompress names scan chains internally. Depending on the design architecture, you must use this internal naming format to identify Tessent TestKompress scan chains in the mask file as follows:

Modular EDT — `edt_<block_name>_channel<number>`

Top-level EDT — `edt_channel<number>`

Use the [report_edt_pins](#) command to determine the number associated with a scan chain.

The mask file must be in a pattern based format. Most ATE’s return accumulated cycle-based failure data. Use the [read_failures](#) command to convert cycle-based failure data to pattern based data for the mask file.

Table 6-17. Mask File Statements

Statement	Usage Rules
<location>	<p>Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows:</p> <p>EDT Off:</p> <ul style="list-style-type: none"> • If “type mask” or “type observe” precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name> <p>EDT On:</p> <ul style="list-style-type: none"> • If “type mask” or “type observe” precedes the location statement, use one of these three formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> <pattern_index> <channel_name> <channel_cell_index> <p>Specify the <i>channel_name</i> argument as follows:</p> <p>For non-modular flows: <i>edt_channel<number></i>. For example, <i>edt_channel3</i>.</p> <p>For modular flows: <i>edt_<block_name>_channel<number></i>. For example, <i>edt_block1_channel3</i>.</p> <p>Note, the <i>channel_cell_index</i> argument refers to the unload cycle.</p> <ul style="list-style-type: none"> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <channel_name> <channel_output_pin_name>

Table 6-17. Mask File Statements (cont.)

Statement	Usage Rules
type <value>	<p>Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values:</p> <ul style="list-style-type: none"> • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for <i>all</i> patterns. Overrides other type statements that cover the same location. <p>Use as many type statements as you like. By default, the tools assume “type mask” for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.</p>

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
// pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
// "type observe" for cell 3 of chain 5 for pattern 40, and
// subsequent "type observe" for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
// pattern 65.
65 chain5 99
```

- **-MASKFILE _pattern_offset number**

An optional switch and integer that specifies the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked.

- **-PREserve _masked _observe _points**

An optional switch, used with external, that allows you to preserve existing masked observation points (wherever Xs appear in a pattern's unload and measure_po values) in external patterns loaded into the tool. When you use this switch to preserve existing masked observation points in the external patterns, the internal patterns stored after simulating the external patterns preserve these masked observation points. You can optionally specify “-Mask maskfile” with additional observation points to be masked.

- **-Store _patterns**

An optional switch that directs the tool to place patterns it simulates during the Good or Fault system mode into the internal pattern set. You can then use the write_patterns command to save these patterns to an external file.

You cannot use this option with the Internal pattern source.

- **-ALL_patterns**

An optional switch that, in analysis mode only, specifies for the tool to add all simulated patterns to the internal pattern set, provided there is at least one undetected fault in the fault list. The simulated patterns can be either randomly generated or can be from the specified external pattern file. Patterns that cause bus conflicts, which the tool ordinarily discards, are included when you use this switch. Without this switch, the tool adds only the effective patterns from a simulation run to the internal pattern set.

- **-NOPadding**

An optional switch specifying that the source test pattern set contains ASCII patterns that are not padded for the scan load and unload data. For example, the source pattern set may be one that you wrote with the write_patterns command using its -NOPadding switch.

You cannot use this option with the Internal pattern source.

Examples

Example 1

The following example performs fault simulation on an external pattern file:

```
set_system_mode analysis
set_pattern_source external file1
add_faults -all
simulate_patterns
report_statistics
```

Example 2

The following example fault grades transition faults using a set of path delay patterns. Assume path delay patterns generated with “`set_fault_type path_delay -mask_nonobservation_points`” are stored in the file, `path_patterns_mask.ascii`:

```
set_system_mode analysis
set_pattern_source external path_patterns_mask.ascii -mask
set_fault_type transition
add_faults -all
simulate_patterns
report_statistics
```

Example 3

The following example shows how to combine patterns from several different files into one pattern set, which you can save in any supported format. First, the example loads patterns from three separate pattern files into the tool’s external pattern set. The patterns are in the same order in which they are loaded. The example then writes the contents of the external pattern set into a new file in the WGL format. Notice the patterns loaded are in mixed formats (ASCII and binary); also, that the tool automatically knows to use the GNU gzip utility when processing the input file with the “.gz” extension.

```
set_pattern_source external my_patterns1.ascii
set_pattern_source external my_patterns2.binary -append
set_pattern_source external my_patterns3.ascii.gz -append
write_patterns my_patterns_all.wgl -wgl -external
```

Example 4

The following commands create a new pattern set using the existing pattern set `pat.ascii` and keep the mask observation points plus the additional masked bit specified in maskfile `my_mask`:

```
set_pattern_source external pat.ascii -store \
    -preserve_masked_observation_points -maskfile my_mask
create_patterns
write_patterns pat_new_masks.ascii
```

Example 5

The following example demonstrates how to specify the offset to speed up the pattern mask flow. Assume that a group of pattern files are created as follows:

Pat1.stil & Pat1.v contain pattern index 0 to 999
 Pat2.stil & Pat2.v contain pattern index 1000 to 1999
 Pat3.stil & Pat3.v contain pattern index 2000 to 2999
 Pat4.stil & Pat4.v contain pattern index 3000 to 3999

After Verilog simulation, only pattern file Pat3.v fails simulation with the following mismatch bit in maskfile Pat3.fail:

```
// This File is simulation generated (Pat3.v)
//format pattern
//failure_buffer_limit_reached none
//data_reference cell
//pattern_id chain/PO_name cell_number expected_value simulated_value
scan_test
    2001           chain1           513           L           H
// reg132/Q
last_pattern_applied      2999
// failing_patterns= 1 simulated_patterns= 1000 simulation_time= 2600000;
failure_file_end
```

The following command allows you to load only the mismatch pattern file and mask the mismatch bits, then save the pattern file without mismatches:

```
set_pattern_source external Pat3.stil -stil -store \
    -mask Pat3.fail -maskfile_pattern_offset 2000
create_patterns
write_patterns Pat3_masked.v -verilog
write_patterns Pat3_masked.stil -stil
```

Related Topics

[write_patterns](#)
[set_abort_limit](#)
[set_capture_clock](#)
[set_observation_point](#)
[set_random_atpg](#)
[set_checkpointing_options](#)

set_pattern_type

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the type of pattern(s) used by ATPG simulation.

Usage

```
set_pattern_type [-RAM_sequential {OFF|ON}] [-MAX_SEQuential {OFF|depth}]
                 [-SEQuential {OFF|depth}] [-MULTiple_load {OFF|ON}] [-CLOCK_po {ON|OFF}]
```

Description

Specifies the type of pattern(s) used by ATPG simulation.

Note

 Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_pattern_type commands until you specify a different setting.

Tip

 If you use ATPG Expert (create_patterns), you do not need to issue this command. ATPG Expert will, for most designs, determine the optimal settings for this command. You may get better results by leaving this command to ATPG Expert.

The set_pattern_type command determines the type of patterns the tool uses during ATPG simulation.

The -Sequential switch provides the ability to use clock sequential cells. For example, “-sequential 2” enables the tool to create clock sequential patterns with up to two clock pulses between scan loads. Try to use the smallest possible depth, as it affects both memory requirements and performance. You can use the [report_sequential_fault_depth](#) command to obtain estimates of the maximum test coverage possible with different sequential depth settings.

When you increase the sequential depth, the tool places all current ATPG untestable faults in the untested fault class. However, multiple-clock pattern compression can be improved by increasing the sequential depth. You cannot decrease the sequential depth when there are any active patterns. You cannot set the depth greater than one if a clock procedure contains a restore_bidis statement since the procedure cannot use sequential ATPG.

If you turn on the RAM sequential option to allow RAM sequential test patterns in the test pattern file, the following rules apply:

1. You may not use RAM sequential patterns with the transition fault type.
2. You may not use RAM sequential patterns with the set_atpg_compression command, but you may use them with the compress_patterns command.

3. You may not use the RAM sequential patterns if a clock procedure contains a `restore_bidis` statement, since the procedure cannot use sequential ATPG.
4. If you are using external patterns that contain RAM sequential patterns, you must set the pattern type to RAM sequential with the `-Ram_sequential` switch.
5. Only RAMs which are proven stable during the load/unload process will be allowed to hold values from one scan load to the next and are testable with RAM sequential patterns.
6. You cannot change the pattern type from RAM or clock sequential while there are any active patterns in the internal or external pattern sets.
7. If you change the pattern type from combinational (the default) to RAM sequential, the tool places all current `atpg_untestable` faults in the `undetected_uncontrolled` class where they are available for additional fault simulation and test generation.
8. You may use failure diagnosis for pattern sets which contain RAM sequential patterns.
9. If you use the `report_gates` command with gate reporting set to `parallel_pattern`, then for the last set of 32 simulated patterns, the tool displays the values for the 2 to 4 vectors of the `ram_sequential` patterns. If you selected a RAM gate, the Report Gate command also displays the internal RAM values.
10. If you use the `write_patterns` command to save RAM sequential patterns, the tool places the argument “`ram_sequential`” on the pattern statement of the ASCII saved patterns.
11. The tool places RAM sequential patterns at the end of the internal pattern set.
12. The fault simulator can detect faults on RAM data lines during RAM sequential patterns, but the test generator will not attempt to create a RAM sequential test for a data line fault. The tool assumes that it can always detect the faults with a non-RAM sequential pattern.
13. Even when you set the pattern type to RAM sequential, the test pattern generator always attempts to find a combinational test first. The test pattern generator only attempts a sequential test generation if all of the following are true:
 - o The test pattern generator identified the fault as组合性不可测的 (combinational ATPG untestable) during the combinational test.
 - o The pattern type is set to RAM sequential.
 - o The fault is connected to an address or write line of an eligible RAM.

The test pattern generator then creates a sequential test depending on how the fault propagates to the RAM. The test pattern generator will only try to create a test that

satisfies one of the following conditions, and if unsuccessful, it will consider the fault to be aborted even if the maximum number of remade decisions has not been exceeded:

- Write Port Address Lines Faults:

Vector 1 - For the first data line of the fault write port, write 0 into the address where the fault address line is at the fault value, and the other address lines are 0 (address A).

Vector 2 - For the first data line of the fault write port, write 1 into the address where the fault address line is at the complements of the fault value, and the other address lines are 0.

Vector 3 - From the first data line of the first read port, read 0 from address A.

- Read Port Address Line Faults:

Vector 1 - For the first data line of the first write port, write 0 into the address where the fault address line is at the fault value, and the other address lines are 0.

Vector 2 - For the first data line of the first write port, write 1 into the address where the fault address line is at the complement of the fault value, and the other address lines are 0 (address A).

Vector 3 - From the first data line of the fault read port, read 1 from address A.

- Write Line Stuck-Off Faults on Multi-Write Port RAMs:

Vector 1 - For the first data line of the first non-fault write port, write 0 into address 0.

Vector 2 - For the first data line of the fault write port, write 1 into address 0

Vector 3 - From the first data line of the first read port, read 1 from address 0.

- Write Line Stuck-On Faults:

Vector 1 - For the first data line of the fault write port, write 1 into address 0.

Vector 2 - For the first data line of the fault write port, write 0 into address 0.

Vector 3 - From the first data line of the first read port, read 0 from address 0 with the first data line of the fault write port at 1.

Arguments

- -RAM_sequential OFF | ON

An optional switch and argument pair that specifies whether or not the test pattern set contains RAM sequential test patterns. This type of pattern is detailed in the “[RAM Sequential Patterns](#)” section of the Tesson Scan and ATPG User’s Manual. This switch is off upon invocation of the tool.

- **-MAX_SEQuential OFF | depth**

An optional switch and argument pair that specifies the maximum clock sequential depth if the tool automatically increases the depth during test generation. (When using the `create_patterns` command to generate test patterns, the tool may increase sequential depth automatically in order to achieve the maximal test coverage.)

When `-max_sequential` is used, ATPG expert can start a lower depth and then later increase up to the max value. It indicates the upper limit, but the tool will start from the optimal. For example:

If the optimal depth is two and you use “`-max_sequential 3`”, the tool will override the setting and start with two.

The Off option (the default) disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number. When using this switch to set maximum clock sequential depth, the depth set by the `-Sequential` switch cannot be greater than the maximum clock sequential depth. The maximum clock sequential depth can be an integer value from 1 to 256.

- **-SEQential OFF | depth**

An optional switch and argument pair that specifies the depth of the design’s non-scan sequential elements (its sequential depth). The integer can be a value between 0 and 255, but the best practice is to set the depth as low as possible because it affects both memory requirements and performance. The default sequential depth upon invocation is OFF which is equivalent to depth zero.

You use the `-sequential` switch most often to specifically set the depth to a specific number. It indicates the sequential start depth even if it is not the optimal number. ATPG Expert will not modify it.

The difference between `-sequential` and `-max_sequential` can be regarded as a pattern generation order strategy. For example:

If the optimal depth is two and you use “`-sequential 3`”, the tool will start with depth of three.

A setting of OFF or 0 enables the tool to create only non-sequential patterns, while a setting of 1 enables the tool also to create basic sequential patterns; with either setting resulting in patterns having a maximum sequential depth of 1. A setting of 2 or more enables clock sequential pattern generation.

Note



Clock sequential patterns must be avoided if a clock procedure contains a `restore_bidis` statement, as the procedure cannot use clock sequential ATPG.

- **-MULTiple_load OFF | ON**

An optional switch and argument pair that allows the tool to create multiple load patterns when clock sequential pattern generation is enabled (the sequential depth is set to 2 or more). This switch is an alternative to `-Ram_Sequential` for testing RAMs if the sequential depth is

set to 3 or 4. Depth 4 is required if the RAM has data hold, H or H1, in its `_cram_read` port definition. Although advisable not to set depth higher than required, it is fine to just set depth to 4 if targeting RAMs with the `-Sequential` option. The multiple load pattern capability is off upon invocation of the tool.

Refer to “[Multiple Load Patterns](#)” in the Tessent Scan and ATPG User’s Manual for additional details about the multiple load pattern type.

- `-CLOCK_po` ON | OFF

An optional switch and argument pair that enables the tool to create clock PO patterns. For a detailed description of this pattern type, refer to “[Clock PO Patterns](#)” in the *Tessent Scan and ATPG User’s Manual*. This switch is on by default.

Note

 Clock PO patterns allow two timings for clock PIs, so an additional timeplate is required to apply these patterns on a tester.

Clock PO patterns cannot be generated if you have defined any non-floating, pulse-always clocks. For more information, see the [add_clocks](#) command.

Examples

Example 1

The following example displays the current pattern type settings:

```
set_pattern_type
// Settings : -RAM_sequential OFF
//           -SEQUENTIAL 0
//           -MAX_SEquential 5
//           -MULTIPLE_load OFF
//           -CLOCK_po      ON
```

Example 2

The following example disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number:

```
set_pattern_type -max_sequential off
// Settings : -RAM_sequential OFF
//           -SEQUENTIAL 0
//           -MULTIPLE_load OFF
//           -CLOCK_po      ON
```

Example 3

The following example creates patterns with up to two clock pulses between scan loads:

```
set_pattern_type -sequential 2
```

```
// Settings : -RAM_sequential OFF
//           -SEQUENTIAL      2
//           -MULTIPLE_load   OFF
//           -CLOCK_po        ON
```

Example 4

The following example enables RAM sequential pattern generation, resets sequential depth to its default setting (0), and disables creation of clock PO patterns.

```
set_pattern_type -ram_sequential on -sequential 0 -clock_po off

// Settings : -RAM_sequential ON
//           -SEQUENTIAL      4/0
//           -MULTIPLE_load   ON/OFF
//           -CLOCK_po        OFF
```

For -SEQUENTIAL, the first number means that the clock sequential depth is 4 and the second number means that the maximal clock sequential depth for clock sequential test cube generation is 0.

For -MULTIPLE_load, the “ON” means that the clock sequential simulation engine allows to simulate test patterns with multiple loads, and “OFF” means that the multiple loads for clock sequential test cube generation is disabled.

Example 5

The following example specifies to create multiple load patterns for testing RAMs. Assuming RAM sequential pattern capability was enabled with a previous set_pattern_type command, the example turns off that capability (recall that the switches retain their settings over multiple commands—until you change them).

```
set_pattern_type -ram_sequential off -sequential 4 -multiple_load on

// Settings : -RAM_sequential OFF
//           -SEQUENTIAL      4
//           -MULTIPLE_load   ON
//           -CLOCK_po        OFF
```

Related Topics

[report_sequential_fault_depth](#)

set_physical_translation

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Translates special Verilog characters to SPICE-readable characters that Calibre applications can read.

Usage

```
set_physical_translation [-Path_divider path_divider] [-Substitute substitute] [-Replacement replacement] [-Global_prefix global_prefix] [-Hier_prefix hier_prefix] [-STrip_prefix strip_prefix] [-Array_delimiters array_delim]
```

Description

Translates special Verilog characters to SPICE-readable characters that Calibre applications can read.

You can restore any of these translation settings to its default setting by issuing the command without the corresponding argument. For specifics for when you use this command, see “[Net Pair Identification with Calibre for Bridge Fault Test Patterns](#)” in the *Tessent Scan and ATPG User’s Manual* for complete information.

Arguments

- **-Path_divider *path_divider***
An optional switch and string specifying the instance path divider. The default is “/X”.
- **-Substitute *substitute***
An optional switch and string for substituting a “#” (pound sign) for a leading character or set of characters for pins, wires, and instance names. By default, the tool substitutes leading “\$”s (dollar signs) with “#”s (pound signs). You can specify a maximum of 10 characters for substitution. If you specify the -Replacement switch and string, then the tool uses this character for the substitution character.
- **-Replacement *replacement***
An optional switch and string replacing the default substitution character. The default is a “#” (pound sign). If you specify a character, then the tool uses this character instead of the default when substituting a leading character or characters for pins, wires, and instance names.
- **-Global_prefix *global_prefix***
An optional switch and string for changing the default global prefix character the tool inserts for pins, wires, and instance names. The default is a “#” (pound sign).
- **-Hier_prefix *hier_prefix***
An optional switch and string changing the hierarchy prefix. The default is “ “ (space).

- **-STrip_prefix *strip_prefix***

An optional switch and string stripping the prefix. This argument is in Calibre form (for example, to strip Xbogus_top/Xmodule/net, you specify -strip_prefix Xbogus_top).

- **-Array_delimiters *array_delim***

An optional switch and string changing the array delimiter. The default is “[]” (square brackets).

set_possible_credit

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the percentage of credit that the tool assigns possible-detected faults.

Usage

`set_possible_credit percentage`

Description

Specifies the percentage of credit that the tool assigns possible-detected faults.

The `set_possible_credit` command specifies the percentage of possible-detected faults that the tool considers as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. For the equations that the tool uses in these calculations, refer to “[Testability Calculations](#)” in the *Tessent Scan and ATPG User’s Manual*.

When you invoke the tool, the default credit value for possible-detected faults is 50 percent. Note that when you set the possible credit to 0, the tool will not change any faults to PT or PU (the existing PT/PU faults will remain).

Arguments

- *percentage*

A required integer, from 0 to 100, that specifies the percentage of possible-detected faults that you want the tool to consider as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. The default value upon invocation of the tool is 50 percent.

Examples

The following example sets the credit for possible detected faults as 25 percent for determining the fault coverage, test coverage, and ATPG effectiveness:

```
set_system_mode analysis
set_possible_credit 25
create_patterns
report_statistics
```

set_power_control

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables low-power ATPG and sets up a switching and/or capture threshold.

Usage

Specify low-power shift and capture

```
set_power_control {OFF|ON} [-SWitching_threshold_percentage percent_integer]
```

Specify low-power shift

```
set_power_control SHIFT {OFF|ON}  
[-SWitching_threshold_percentage percent_integer]  
[-REjection_threshold_percentage {percent_integer | OFF}]
```

Specify low-power capture

```
set_power_control CAPTURE {OFF|ON}  
[-SWitching_threshold_percentage percent_integer]  
[-REjection_threshold_percentage {percent_integer | ST_Peak | ST_Average | WSA_Peak  
| WSA_Average} | OFF}]
```

Hybrid TK/LBIST Flow Usage

```
set_power_control shift {off | on} [-switching_threshold_percentage percent_integer]
```

Description

Enables low-power ATPG and sets up a switching and/or capture threshold.

When the command is entered without any arguments, the current power control settings display.

Note

 When you use set_power_control, there is an impact on the ATPG test pattern count to satisfy the power constraints. There is also an impact on the tool run time because of the extra effort required to compute the switching activity for every test pattern.

For more information, see the [Low-Power ATPG](#) section in the *Tessent Scan and ATPG User's Manual*

Note

 When the [set_edt_power_controller](#) command is set to *Enabled*, the default for the set_power_control SHIFT command is ON for TestKompress only.

Note

 If during EDT IP creation, EDT lower-power controller is not enabled, but during ATPG, “set_power_control shift on” is used, the tool issues this message:

```
// Error: Shift power control is requested, but power control hardware  
// is not present or is disabled.  
// Please check the "set_edt_power_controller shift" command.
```

Note

 If during EDT IP creation, EDT lower-power controller is enabled, and during ATPG, “set_power_control shift on” is used, but they are not in the same bias range, the tool issues this message, for example:

```
// Warning: Specified software switching threshold [5] is not consistent  
// with the switching threshold used to generate the shift power  
// control hardware [20] in block "blk1".  
// The software and hardware thresholds should be in the same  
// bias range (except for the full control case).  
// The following are the valid bias ranges: [0, 11], [12, 24],  
// [25, 50].
```

For more information on using low-power ATPG with Tesson TestKompress, see “[Low-Power Test](#)” in the *Tesson TestKompress User’s Manual*.

Hybrid TK/LBIST Flow Specifics

Specifies the switching threshold in the patterns for fault simulation.

You can only use the low-power shift usage of this command for logicBIST, otherwise, the tool issues an error.

Note

 You cannot use the command’s existing -rejection_threshold_percentage switch with the low-power logicBIST controller.

Arguments

- CAPTURE OFF|ON

Optional literal and switch that enables low-power capture. Options include:

OFF—Disables low-power capture. Default.

ON—Enables low-power capture.

- SHIFT OFF|ON

Optional literal and switch that enables low-power shift. Options include:

OFF—Disables low-power shift. Default.

If the [set_edt_power_controller](#) command is set to *Enabled*, the *Off* option of `set_power_control` shift sets all masking bits to 1, which enables all internal chains; however, it does not disable low-power TestKompress hardware. To disable low-power TestKompress hardware and bypass the low power registers, you must force the `edt_low_power_shift_en` signal off in the test procedure file and execute the [set_edt_power_controller](#) *Disabled* command and option.

`ON` — Enables low-power shift.

- `-SWitching_threshold_percentage percent_integer`

Optional switch and integer pair that specifies the switching threshold for low-power ATPG. The percentage integer can be 1 to 100. By default, for Tessent TestKompress applications, the switching threshold for ATPG is set to match the maximum switching threshold defined for the power controller hardware if that threshold is equal to or greater than 15%; if maximum hardware switching is less than 15%, the software default is 15%. Note, maximum hardware switching is specified by `set_edt_power_controller Shift -MIN_Switching_threshold_percentage percentage_integer`.

For more information, see “[Low-Power Test](#)” in the *Tessent TestKompress User’s Manual*.

- `-switching_threshold_percentage percent_integer (Hybrid TK/LBIST Flow Only)`

An optional switch and integer pair that specifies the maximum switching activity during shift. The *percent_integer* value can be between 1 and 50.

The tool runs fault simulation with the appropriate control register values to implement the specified switching threshold.

When saving the tester format patterns in Tessent Shell, the registers are seeded with the same values used during pattern simulation. It is not possible to save hardware default mode tester patterns when the switching threshold specified during fault simulation is different from the switching threshold specified during IP creation. An error message will be issued if this switching threshold cannot be supported by the low-power hardware.

Refer to the [Hybrid TK/LBIST Flow User’s Manual](#) for complete information.

- `[-REjection_threshold_percentage {percent_integer | OFF}]`

Optional switch and integer pair that specifies a state element switching limit per test pattern. Test patterns that exceed the specified switching limit are discarded. By default, this option is off and if rejecting/discardng test patterns that exceed the threshold has a negative effect on test coverage, the specified threshold is violated.

- `[ST_Peak | ST_Average | WSA_Peak | WSA_Average]`

Optional literal that filters the test pattern rejection threshold based on a specified metric. When a specified metric exceeds the rejection threshold, the pattern is discarded. These options can only be used with the low-power capture feature. Literals include:

`ST_Peak` — Peak of state transitions. Default.

`ST_Average` — Average of state transitions.

`WSA_Peak` — Peak of weighted switching activity.

WSA_Average — Average of weighted switching activity.

Examples

Example 1

The following example sets low-power capture with a 30% threshold and low-power shift with a 20% switching threshold and generates test patterns:

```
set_power_control capture on -switching_threshold_percentage 30 \
    -rejection_threshold_percentage 35
set_power_control shift on -switching_threshold_percentage 20 \
    -rejection_threshold_percentage 25
create_patterns
```

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold and enabled for test pattern generation.

```
set_edt_power_controller shift -min_switching_threshold_percentage 20
...
write_edt_files low_power_enabled_edt.v
```

Given the compression logic created with the preceding command, the following commands are used during test pattern generation to enable the controller with a 22% switching threshold for shift and generate test patterns.

```
set_power_control shift on -switching_threshold_percentage 22
set_system_mode analysis
create_patterns
```

Example 3 (Hybrid TK/LBIST Flow Only)

This example configures the low-power logicBIST logic to target a switching threshold of 10%. The actual values loaded in the low-power control registers to achieve this switching threshold can be seen using the report_edt_configuration command. The power metrics can be seen using the report_power_metrics command.

```
set_power_control shift on -switching 15
set_system_mode analysis
report_edt_configuration
```

```

// IP version: 5
// Shift cycles: 128, 120 (internal scan length) + 8
// (additional cycles)
// External scan channels: 4
// Internal scan chains: 480
// Masking bits: 53
// Longest chain range: 2 - 120
// Decompressor size: 32
// Compactor type: Xpress
// Scan cells: 57600
// Compression per pattern: 112.50x (ATPG bypass = 4 x 14400,
// EDT = 4 x 128)
// Clocking: edge-sensitive
// Compactor pipelining: Off
// Chain mask: On (load 0, unload 0)
// LBIST: On
// MISR compactor pipelines: 0
// PRPG seed: 00000001
// Low power shift controller: Enabled and active
// Min switching threshold: 15%
// Hold value: 1010
// Toggle value: 1011
// Switching control: 0011
// 
```

report_power_metrics -shift

```

// Note: Pattern classification is automatically turned off.
Power Metrics      Min.    Average   Max.
-----
Load Shift Transitions  4.45%  14.26%  23.51%
Response Shift Transitions 42.48% 49.16% 50.04%

```

Related Topics

[report_edt_configurations](#)
[set_edt_power_controller](#)
[report_power_metrics](#)

set_power_metrics

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Calculates shift and capture power during test pattern generation and simulation.

Usage

```
set_power_metrics [-SHIft {OFF|ON}] [-CAPture {OFF|ON}]  
[-COMPosite_shift_power {OFF|ON}] [-X_CAPture_transition_credit float]  
[-X_SHIft_transition_credit float]
```

Description

Calculates shift and capture power during test pattern generation and simulation.

Shift power is the test power consumed during the loading of test stimuli and the unloading of responses. Capture power is the test power consumed by switching activity in the circuit during capture cycles.

Note

 When you use set_power_metrics, there is an impact to the tool run time during create_patterns and simulate patterns, because of the extra effort required to compute the switching activity for every test pattern.

Arguments

- **-SHIft OFF|ON**
An optional switch and literal pair that enables the calculation of the power consumed for switching activity in the circuit during shift.
- **-CAPture OFF|ON**
An optional switch and literal pair that enables calculation of the power consumed for switching activity in the circuit during capture cycles.
- **-COMPosite_shift_power OFF|ON**
An optional switch and literal pair that combines the load and unload switching activity in shift power calculations. By default, the load and unload switching activity is computed separately.
- **-X_CAPture_transition_credit *float***
An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x capture transitions when calculating switching activity in capture cycles (the default values are 0.1, 0.05, 0.05, respectively). The specified float value must be between 0 and 1, and it applies to all three x values.

- **-X_SHIfT_transition_credit float**

An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x shift transitions when calculating switching activity during shift.

The specified value applies to all x values. The float number should be between 0 and 1. The default is 0.5.

Examples

Example 1

The following example calculates shift power and capture power during test pattern generation, fault simulation, and good simulation and displays the capture power for capture cycles and the shift power for shift transitions.

```
set_power_metrics -shift on -capture on
create_patterns
```

```
Statistics report
-----
#faults      #faults
fault class      (coll.)      (total)
-----
FU (full)        70256       88262
-----
UC (uncontrolled)    92         118
UO (unobserved)    482        626
DS (det_simulation) 35006      44416
DI (det_implementation) 17321      22589
PU (posdet_untestable) 46          62
PT (posdet_testable) 11          23
UU (unused)        134         134
TI (tied)          94          98
RE (redundant)     240         265
AU (atpg_untestable) 16830      19931
-----
test_coverage     75.02%      76.39%
fault_coverage    74.52%      75.96%
atpg_effectiveness 99.17%      99.14%
-----
#test_patterns      494
#clock_sequential_patterns 494
#simulated_patterns 512
CPU_time (secs)      32.4
-----
Power Metrics      Min.      Average      Max.
-----
WSA                9.03%      18.74%      36.82%
State Element Transitions 8.99%      25.44%      47.33%
-----
Load Shift Transitions 39.06%      44.62%      51.66%
Response Shift Transitions 6.45%      39.38%      50.54%
```

For Power Metrics:

- The WSA (Weighted Switching Activity) lists the minimum, average, and maximum percentage of the test power consumed by the switching activity of the combinational logic and state elements during capture cycles.
- The State Element Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of state element transitions during capture cycles.
- The Load Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of scan cell transitions during the loading of test stimuli.
- The Response Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of the scan cell transitions during the unloading of responses.

Example 2

The following example reports the power metrics for external test patterns, *mypat.ascii*, with the composite shift power metric on and then off:

```
read_patterns mypat.ascii
set_power_metrics -composite on
report_power_metrics -external

Power Metrics          Min.    Average   Max.
-----
WSA                  10.62%  12.54%  13.65%
State Element Transitions  13.59%  18.71%  20.94%
-----
Peak Cycle
-----
WSA                  10.62%  12.54%  13.65%
State Element Transitions  13.59%  18.71%  20.94%
-----
Load Shift Transitions  15.38%  25.00%  30.77%
Response Shift Transitions  17.95%  21.15%  25.64%
Composite Shift Transitions  13.24%  20.08%  26.69%
```

```
set_power_metrics -composite off
report_power_metrics -external
```

Power Metrics	Min.	Average	Max.
WSA	10.62%	12.54%	13.65%
State Element Transitions	13.59%	18.71%	20.94%
Peak Cycle			
WSA	10.62%	12.54%	13.65%
State Element Transitions	13.59%	18.71%	20.94%
Load Shift Transitions	15.38%	25.00%	30.77%
Response Shift Transitions	17.95%	21.15%	25.64%

Example 3

The following example sets up power metric calculations to use composite shift, generates test patterns, and reports shift power for all the patterns:

```
set_power_metrics -composite_shift on
create_patterns
report_power_metrics -shift -patterns all

// Pattern Shift
// Pat. loaded Pat. unloadedAverage
// -----
// 0      -      10.60%
// 1      0      8.60%
// 2      1      14.20%
// 3      2      8.20%
// pad    3      9.56%
```

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests:

```
set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all
```

```
// Chain test :  
//  
//      Pattern      Shift  
// Pat. loaded Pat. unloadedAverage  
// ----- ----- -----  
// 0      -          32.89%  
// 1      0          48.60%  
// 2      1          54.20%  
// pad   2          49.56%  
//  
// Scan test :  
//  
//      Pattern      Shift  
// Pat. loaded Pat. unloadedAverage  
// ----- ----- -----  
// 0      -          10.60%  
// 1      0          8.60%  
// 2      1          14.20%  
// 3      2          8.20%  
// pad   3          9.56%
```

Related Topics

[report_power_metrics](#)

[set_power_control](#)

set_procedure_cycle_checking

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.

Usage

`set_procedure_cycle_checking ON | OFF`

Description

Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.

This command helps detect timing problems in test procedures earlier on in the ATPG process. By default, the test procedure cycle timing checking is performed after scan chain tracing. If an error condition is detected, the tool remains in the Setup mode. You can then modify the test procedures and reissue the `set_system_mode` command.

Arguments

- **ON**

A literal that specifies for the tool to `set_procedure_cycle_checking ON`. This is the default behavior upon invocation of the tool.

- **OFF**

A literal that specifies for the tool to `set_procedure_cycle_checking OFF`. In order to turn procedure cycle checking off, you must be in Setup mode.

Examples

```
set_system_mode setup
set_procedure_cycle_checking OFF
```

Related Topics

[set_system_mode](#)

set_procedure_retargeting_options

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.

Usage

```
set_procedure_retargeting_options [-scan {ON | OFF}]  
[-ijtag {ON | OFF}]  
[-timeplate timeplate_name]  
[-pulse_during_shift clock_port_list]
```

Description

Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.

By default, the tool automatically retargets core-level test_setup and test_end IJTAG procedures, and adds the needed iCalls to the test_setup or test_end procedure. If you disable the -ijtag behavior by setting this switch to off, you will need to manually add all core level iCalls.

In patterns -scan context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level and merges the retargeted core-level procedures with top-level procedures if top-level procedures exist.

In patterns -scan_retargeting context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level if chip-level load_unload and shift test procedures do not exist.

If you disable the -scan behavior by setting it to off, and the chip-level load_unload and shift test procedures are missing from the top-level test procedure, the tool generates an error.

Arguments

- **-scan {ON | OFF}**

A literal that specifies to the tool whether to enable test procedure retargeting. By default upon invocation, test procedure retargeting is enabled. In order to disable test procedure retargeting, you must be in Setup mode. For more information on test procedure retargeting, see “Core Mapping for ATPG Process Overview” in the *Tessent Scan and ATPG User’s Manual*.

- **-ijtag {On | Off}**

An optional switch that specifies whether automatic mapping of core-level IJTAG test_setup and test_end procedures to the next higher level of hierarchy is enabled. You should use the Off switch to disable automatic IJTAG mapping if you are manually generating the entire top-level test_setup and test_end procedures.

- **-timeplate *timeplate_name***

An optional switch that specifies the timeplate to be used in the generated load_unload and shift procedures. The tool determines the timeplate to use as follows:

- a. Uses the timeplate specified by the -timeplate switch if one is specified.
- b. If one is not specified:
 - In patterns -scan context, if the top-level load_unload and shift procedures are present, the tool uses the timeplate from the top-level procedure.
 - In patterns -scan_retargeting context or if top-level procedures are not provided, the tool uses the default test_setup timeplate.
- c. If neither a tool-generated or user-defined test_setup procedure exists, the tool uses the default timeplate. For more information on the tool-generated test_setup procedure, see the description in “[Test Procedure Retargeting for Scan Pattern Retargeting](#)” in the *Tessent Scan and ATPG User’s Manual*.
- d. If the default timeplate is not specified, the tools uses the first timeplate read in.

- **-pulse_during_shift *clock_port_list***

An optional switch that specifies to pulse the clocks specified by the *clock_port_list* argument during the shift procedure. The *clock_port_list* argument can be a Tcl list of clocks or gate_pins that are primary inputs. To reset this specification, re-invoke this option and provide an empty list.

This option is typically used to specify clocks that drive top-level pipeline stages outside the core. The tool can map the load_unload and shift procedures from the core(s) to the top. However, if top-level pipeline stages are clocked by separate clocks, the tool will not know to pulse them, and, as a result, will be unable to trace the scan channels from the cores to the top (R2 DRC violations). This option instructs the tool to additionally pulse the specified clock(s) in the created top-level shift procedure.

Examples

Example 1

The following example specifies a timeplate to use in the load_unload and shift procedures.

```
set_procedure_retargeting_options -timeplate gen_tp1
```

Example 2

The following example specifies to pulse two clocks.

set_procedure_retargeting_options -pulse_during_shift {top_clock pipe_clock}

Related Topics

[set_procfile_name](#)

[set_procfile_name](#)

Context: all contexts

Mode: setup

Specifies a new procedure file for the tool to process at a later time.

Usage

```
set_procfile_name procedure_file_name [-override_scangroup_proc]
```

Description

Specifies a new procedure file for the tool to process at a later time.

The `set_procfile_name` command reads the specified procedure file and performs basic syntax checking. The tool reports errors it finds but does not process and store the file at the time you issue the command.

Note

 The procfile may contain Tcl variable that will be substituted when the file is read. The referenced Tcl variable must exist in the Tcl global namespace scope. If you are setting the variables within a proc, make sure to use `::` as a prefix to the variable name such that it is set in the global namespace. For example, use “`set ::my_period 4`” such that `$period` exists when processing the proc files. For more info about Tcl variable substitution inside the procfile, refer to “[Additional Support for Test Procedure Files](#)” in the *Tessent Shell User’s Manual*.

You can specify only one procedure file with this command. If you issue the command again, the new procedure file overrides the one previously set.

If procedure files with scan groups exist, the tool processes the scan group procedure files first, followed by the procedure file specified in this command. You can override the scan group procedure files and only process *procedure_file_name* by specifying the `-override_scangroup_proc` option.

For scan pattern retargeting, specifies the top-level test procedure file to be used for saving top-level patterns. In this case, the following procedures are expected to be defined in this file: `test_setup`, `load_unload`, `shift`, and `external_capture`.

For core mapping for ATPG and pattern retargeting, you must provide the chip-level test procedure file; the test procedure file must minimally include the timeplate definition. You do not need to provide chip-level `load_unload` and `shift` test procedures; if they do not exist, the tool automatically generates them based on the information in the core-level test procedure files. This step merges the core test procedures if there is more than one core, and maps them to the top-level test procedures. You may also need to specify the `test_setup` procedure for the design, if you need to set up the chip. All other procedures will be automatically mapped by the tool if they are not specified.

For techniques to handle slow scan enable transitions, see “[Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions](#)” in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***procedure_file_name***

A required string that specifies the pathname to the procedure file.

You can use the tab completion method to list possible files for *procedure_file_name* that exist in the current or specified directory. For example, specifying “`set_procfile_name`” and then pressing the Tab key will list the files in the current directory. Specifying “`set_procfile_name ..//data/test_pr`” and pressing the Tab key will list the files with names starting with `test_pr` in the `..//data` directory.

- **-override_scangroup_proc**

An optional argument that causes the tool to disregard scan group procedure files and only process *procedure_file_name*.

Examples

Example 1

The following example shows how the `set_procfile_name` command is used in an IJTAG retargeting flow.

```
set_context patterns -ijtag -no_rtl
...
set_procfile_name data/setup.proc # initial syntax checking done
set_system_mode analysis # procedure file opened and parsed here
```

Example 2

The following example shows how to override the scan group procedure files.

```
set_procfile_name data/setup.proc -override_scangroup_proc
```

Example 3

The “[Retargeting Example](#)” in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

set_quick_synthesis_options

Context: unspecified, dft, patterns

Mode: setup

This command sets options for quick synthesis.

Usage

```
set_quick_synthesis_options [-Parallel_synthesis {maxcpu | integer}]
                           [-Skip_synthesis_of_two_dimensional_arrays_larger_than {unlimited | integer}]
                           [-max_for_loop_size integer] [-blackbox_design_ware_modules {on | off}]
                           [-verbose {on | off}]
```

Description

The tool performs quick synthesis, if it is needed, during check_design_rules, extract_icl, or when switching to the analysis system mode. Quick synthesis creates new instance, port, pin, and net objects to implement RTL constructs so that they are usable by analysis. Objects created during quick synthesis have their created_by_synthesis attribute set to true.

Note

 You can report modules or instances that are synthesized using the `is_synthetized` [instance](#) or [module](#) attributes, respectively. For example:

```
get_module -filter is_synthetized
```

Arguments

- `-Parallel_synthesis {maxcpu | integer}`

An optional switch and literal or integer pair that sets the number of parallel processes for the quick synthesis. When specified with maxcpu, the number of parallel jobs will be the number of available CPUs. When specified with an *integer*, the number of parallel jobs is the specified number. You should use a number less than or equal to the number of available CPUs, otherwise the tool issues a warning. The default is four CPUs.

- `-Skip_synthesis_of_two_dimensional_arrays_larger_than {unlimited | integer}`

An optional switch and literal or integer pair that specifies the bit-count threshold for the compilation of memories in a module. If this total bit-count is more than the specified threshold, the module is not compiled and is treated as a black box. The default setting is 4096 bits.

The purpose of this feature is to avoid synthesizing memory models for which you have loaded the behavioral model in Tessent shell. If you have large multi-dimensional arrays in your RTL and you want to synthesize them, specify the `add_black_boxes -module module` command on all behavioral memory models and enlarge this threshold to match your requirement. Explicitly black-boxing the memory modules will exclude them from synthesis. You can also set the `exclude_from_synthesis` attribute on the memory modules if

you don't want to black box them but only prevent them from being synthesized. During quick synthesis, modules which were automatically black boxed because they contained a multi-dimensional array exceeding the threshold value are listed in a warning similar to the following:

```
// Warning: The following modules were not synthesized because they
// include multi-dimensional arrays whose bit count exceeds the
// following limit: 4096. You can increase this limit with command:
// 'set_quick_synthesis_option
// -skip_synthesis_of_two_dimensional_arrays_larger_than <int>'
// but you should first check that no memory module is listed below.
// 'SYNC_1RW_32x256_RC_BISR'
// To avoid synthesizing behavioral Verilog views of memories, you
// need to provide:
// - their tesson library description, or;
// - their verilog model along with their tcd_memory_lib
// description;
// If you run logictest DRCs, and do not have a tesson library
// description for your memory, you must provide a Verilog model
// which hides the memory core for all memory instances that have
// embedded test logic. Finally, you can add an explicit black box
// declaration for all memory instances if nothing in
// their models needs to be visible for the logictest DRC.
```

- **-max_for_loop_size *integer***

An optional switch-value pair that prevents quick synthesis on modules containing for loops that are larger than the specified *integer* parameter. The default is 5000.

- **-blackbox_design_ware_modules on | off**

An optional switch and literal pair that controls whether the tool automatically blackboxes DesignWare modules that contain no definitions. The default is off.

- **-verbose on | off**

An optional switch that writes extra information messages to stdout or the tool's log file. The default is off.

Examples

Example 1

The following example sets the number of parallel quick synthesis processes to 3:

```
set_quick_synthesis_options -parallel_synthesis 3
```

Example 2

The following example sets a new limit to avoid synthesis for modules with large multi-dimensional arrays to 128:

```
set_quick_synthesis_options -skip_synthesis_of_two_dimensional_arrays_larger_than 128
```

Example 3

The following example enables synthesis for modules with multi-dimensional arrays of any size by setting the threshold to unlimited:

```
set_quick_synthesis_options \
    -skip_synthesis_of_two_dimensional_arrays_larger_than unlimited
```

Related Topics

[Instance](#)

[Port](#)

[Net](#)

set_ram_initialization

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies whether to initialize RAM and ROM gates that do not have initialization files.

Usage

set_ram_initialization Uninitialized | Random

Description

Specifies whether to initialize RAM and ROM gates that do not have initialization files.

The **set_ram_initialization** command allows the tool to internally generate random values and place them into all uninitialized RAM and ROM gates. This command is useful when simulating random patterns.

Arguments

- **Uninitialized**

A literal that specifies for the tool to use unknown (X) values to set the memory elements of all RAM and ROM gates which do not have an initialization file. This is the default behavior upon invocation of Tesson FastScan and Tesson TestKompress.

- **Random**

A literal that specifies for the tool to use random values to set the memory elements of all RAM and ROM gates which do not have an initialization file.

Examples

The following example places random values into all uninitialized RAM and ROM gates:

```
set_ram_initialization random
set_system_mode analysis
create_patterns
```

Related Topics

[read_modelfile](#)

[write_modelfile](#)

[set_random_atpg](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool uses random patterns during ATPG.

Usage

`set_random_atpg ON | OFF`

Description

Specifies whether the tool uses random patterns during ATPG.

The `set_random_atpg` command controls whether the tool uses random test generation techniques to create_patterns during the ATPG process.

Arguments

- **ON**
A literal that specifies for the tool to use random patterns to create test patterns. This is the default behavior upon invocation of the tool.
- **OFF**
A literal that specifies for the tool not to use random patterns to create test patterns. When you use this option, the tool only performs deterministic ATPG.

Examples

The following example turns off the random ATPG process, so only the deterministic ATPG is performed:

```
set_system_mode analysis  
set_random_atpg off  
create_patterns
```

set_random_clocks

Context: dft -edt, patterns -scan

Mode: analysis

Specifies whether Tesson FastScan uses combinational or clock_sequential patterns for random pattern simulation.

Usage

```
set_random_clocks {pin_name | pin_pathname} ...
```

Description

Specifies whether Tesson FastScan uses combinational or clock_sequential patterns for random pattern simulation.

The set_random_clocks command specifies for Tesson FastScan to either use combinational random patterns or use the pins that you specify for clock_sequential random patterns during simulation.

Combinational random pattern simulation is the default upon invocation. You specify clock_sequential by entering the set_random_clocks command with an ordered set of clock/read/write lines which Tesson FastScan exercises in the same order during the clock_sequential random pattern simulation.

You can reset Tesson FastScan to its invocation default of combinational random pattern simulation by entering the set_random_clocks command without any arguments. Also, when you re-enter the Setup system mode, Tesson FastScan deletes the random clock list and is once again ready for combinational random pattern simulation.

Arguments

- *pin_name* | *pin_pathname*

A required repeatable string that specifies the names of defined clock, read, and write lines. You must list these pins in the order in which you want Tesson FastScan to exercise them during the clock_sequential random pattern simulation. The default is none, which specifies for Tesson FastScan to perform combinational random pattern simulation.

Tesson FastScan displays an error if any specified pin is not a defined clock, read, or write line. The tool also displays an error if the number of pins that you list is equal to or exceeds the selected sequential depth.

Examples

The following example runs random pattern simulation with clock_sequential patterns:

```
set_pattern_type -sequential 10
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 sk1 sk2
set_system_mode analysis
set_random_clocks sk1
add_faults -all
simulate_patterns -source random
```

Related Topics

[set_capture_clock](#)
[simulate_patterns](#)

set_random_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the number of random patterns that the tool simulates.

Usage

`set_random_patterns integer`

Description

Specifies the number of random patterns that the tool simulates.

The `set_random_patterns` command specifies how many random patterns you want the tool to simulate.

Arguments

- *integer*

A required integer, greater than or equal to 0, that specifies the number of random patterns you want the tool to simulate. The invocation default is 1024.

Examples

The following example sets the number of random patterns to analyze its controllability effects:

```
set_system_mode analysis
set_random_patterns 612
```

Related Topics

[set_capture_clock](#)

[set_observation_point](#)

[simulate_patterns](#)

set_read_verilog_options

Context: unspecified, all contexts

Mode: all modes

Sets certain options globally for subsequent `read_verilog` commands.

Usage

```
set_read_verilog_options [-allow_enum_relaxation {on | off}]
```

Description

Sets certain options globally for use in subsequent `read_verilog` commands.

Arguments

- `-allow_enum_relaxation on | off`

An optional switch and literal pair that, when set to “on”, instructs the tool to bypass specified LRM Verilog rules. The default is off.

The IEEE standard requires that when a driver is assigned to a variable of an enum type, the driver must itself be declared to be a variable of the same type or to be a value that lies within the enumeration set defined for that type unless an explicit cast is used or unless the enum variable is a member of a union. See section 4.10.3 in IEEE 1800-2005.

If you want to relax this rule, then set this option to “on”.

set_register_value

Context: all contexts

Mode: all modes

Allows you to specify the register value variable outside of Setup mode.

Usage

```
set_register_value value_name value_string  
[-RAdix {BInary | HEx | OCtal | Decimal}]
```

Description

Allows you to specify the register value variable outside of Setup mode.

When using this command, you must have previously specified the [add_register_value](#) command with the -Width command as well as the required arguments.

See “Serial Register Load and Unload for LogicBIST and ATPG” in the *Tessent Shell User’s Manual* for complete information.

Arguments

- ***value_name***
A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.
 - ***value_string***
A required state string whose default radix is binary. You can specify Z or X values in addition to numerical values appropriate for the radix.
 - -Radix {**Binary** | Decimal | Octal | Hexadecimal}
An optional switch and literal that specifies a different radix.

Examples

The following example deletes all register value variables, specifies a register value variable with a decimal radix and register width of 37, and then sets the value of the register.

Related Topics

[add_register_value](#)
[report_register_value](#)
[delete_register_value](#)

set_relevant_coverage

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Determines which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.

Usage

```
set_relevant_coverage [{-Include | -Exclude} {fault_class[fault_subclass] ...}]
```

Description

Determines which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.

Issue this command without any arguments to report the current settings.

Arguments

- **-Include**

An optional switch that includes the specified fault sub-class(es) in the relevant coverage reported by the tool.

- **-Exclude**

An optional switch that excludes the specified fault sub-class(es) from the relevant coverage reported by the tool.

- ***fault_class[fault_subclass]***

An optional, repeatable string that specifies the fault class or fault class/sub-class to include or exclude. [Table 6-18](#) lists the fault classes that you can specify with this command and whether they are included in the reported relevant coverage by default. [Table 6-19](#) lists the available fault sub-classes. You can specify a fault class or sub-class either by name or code. For AU faults, you can define a customized sub-class using a unique name and instruct the tool to include (default) or exclude this user-defined sub-class from the relevant fault coverage. For information about using a user-defined sub-calls, refer to “[Example 2](#)” on page 2348.

Table 6-18. Fault Classes for Relevant Coverage

Name	Code	Included in Relevant Coverage by Default?
ATPG_UNTESTABLE	AU	yes
REDUNDANT	RE	no
TIED	TI	no
BLOCKED	BL	no
UNUSED	UU	no

Table 6-19. Fault Sub-Classes for Relevant Coverage

Fault Class	Fault Sub-Class	Code	Included in Relevant Coverage by Default?
AU	BLACK_BOXES	BB	yes
	CELL_CONSTRAINTS	CC	yes
	EDT_BLOCKS	EDT	no
	FALSE_PATHS	FP	no
	HOLD_PI	HPI	yes
	HYBRID_LBIST	LBIST	no
	IJTAG	IJTAG	no
	LOW_PIN_COUNT_TEST	LPCT	no
	MASK_PO	MPO	yes
	MULTICYCLE_PATHS	MCP	yes
	ON_CHIP_CLOCK_CONTROL	OCC	no
	PIN_CONSTRAINTS	PC	yes
	SEQUENTIAL_DEPTH	SEQ	yes
	TIED_CELLS	TC	yes
	UNDRIVEN	UDN	yes
UU	USER-DEFINED	none	yes
	WIRE	WIRE	yes
UU	CONNECTED	CON	no
	FLOATING	FL	no

Examples

Example 1

The following example removes the PIN_CONSTRAINTS and CELL_CONSTRAINTS sub-classes from relevant coverage reported by the tool:

```
set_relevant_coverage -exclude AU.PC AU.CC  
set_relevant_coverage
```

```
Relevant Coverage Setting Report
-----
Fault Sub-class Relevant Coverage
-----
AU.UDN           yes
AU.BB            yes
AU.WIRE          yes
AU.EDT           no
AU.PC          no
AU.TC            yes
AU.CC          no
AU.FP            no
AU.MCP           yes
AU.HPI           yes
AU.MPO           yes
AU.SEQ            yes
AU.LBIST          no
AU.OCC            no
AU.IJTAG          no
AU.LPCT           no
UU.FL             no
UU.CON            no
TI                no
BL                no
RE                no
-----
```

Example 2

The following example shows how to use a user-defined fault sub-class to exclude specific pins from reporting. This example is based on the following fault list contained in *flist.au.usr1*:

```
//  type   code   pin_pathname
//  ----  -----
0   AU.USR1    /si7
0   EQ.USR1    /sf_1_7/SI
1   AU.USR1    /si7
1   EQ.USR1    /sf_1_7/SI
0   AU.USR1    /si8
0   EQ.USR1    /sf_1_8/SI
1   AU.USR1    /si8
1   EQ.USR1    /sf_1_8/SI
```

This fault list creates a user-defined sub-class for AU faults on two pins for both SA0 and SA1.

```
read_faults flist.au.usr1 -retain
set_relevant_coverage -exclude AU.USR1
report_statistics
```

Statistics Report Stuck-at Faults			
Fault Classes	#faults (total)	#faults (total relevant)	
FU (full)	1397	1389	
UC (uncontrolled)	27 (1.93%)	same (1.94%)	
DS (det_simulation)	728 (52.11%)	same (52.41%)	
DI (det_implication)	315 (22.55%)	same (22.68%)	
UU (unused)	234 (16.75%)	same (16.85%)	
AU (atpg_untestable)	93 (6.66%)	85 (6.12%)	
<hr/>			
Fault Sub-classes			
AU (atpg_untestable)			
BB (black_boxes)	12 (0.86%)	same (0.86%)	
PC* (pin_constraints)	20 (1.43%)	same (1.44%)	
TC* (tied_cells)	53 (3.79%)	same (3.82%)	
USR1	8 (0.57%)	deleted	
*Use "report_statistics -detailed_analysis" for details.			
<hr/>			
Coverage			
test_coverage	89.68%	90.30%	
fault_coverage	74.66%	75.09%	
atpg_effectiveness	98.07%	98.06%	

Related Topics

[report_statistics](#)

[set_fault_subclass_analysis](#)

set_run_synthesis_options

Context: unspecified, all contexts

Mode: all modes

Specifies the default values for options that can be set for the run_synthesis command.

Usage

```
set_run_synthesis_options [-synthesis_output_directory directory]  
  [synthesis_tool [-compilation_options option_list]  
   [-pre_compilation_commands command_list]  
   [-post_compilation_commands command_list] [-startup_file file_name]  
   [-command_name command_name_path]]
```

Description

Specifies the default values for options that can be set for [run_synthesis](#) command. The options that are defined with set_run_synthesis_options are overridden if the are explicitly defined with the run_synthesis command. When this command is used without any options specified, all options are reset to Tesson Shell default values.

Arguments

- **-synthesis_output_directory *directory***

An optional switch and value pair that specifies the synthesis output directory. The default directory is *synthesis_outdir* and is located in the Tesson Shell invocation directory.

- ***synthesis_tools***

A required value that specifies the synthesis tool you wish to use for synthesis when setting the remaining options. The only permitted value is “dc_shell”. Other synthesis tools may be supported in future releases.

- **-compilation_options *option_list***

An optional switch and option list that specifies additional options to be added to the synthesis compile command used in the <*synthesis_tool*>.synthesis_tcl. An example would be the -boundary_optimization option for dc_shell.

- **-pre_compilation_commands *command_list***

An optional switch and command list that adds the provided commands to the <*synthesis_tool*>.synthesis_tcl so that they are executed before the compilation command is run.

- **-post_compilation_commands *command_list***

An optional switch and command list that adds the provided commands to the <*synthesis_tool*>.synthesis_tcl so that they are executed after the compilation command is run.

- **-startup_file *file_name***

An optional switch and file path that allows the user to explicitly specify the startup file that will be used by the synthesis tool. The file specified is copied into the actual synthesis directory.

Note

 If a startup file is not specified and a *.synopsys_dc.setup* file exists in the Tessent run directory, it will automatically be copied into the synthesis run directory. If relative paths are utilized within this setup file, it should be noted the synthesis run directory is two levels below where Tessent Shell is launched and these paths should account for this execution point.

- **-command_name *command_name_path***

Defines the command name to invoke the synthesis tool. The default for *dc_shell* is '*dc_shell*' and the command is assumed to be available in the search path of the operating system.

Examples

The following examples of *set_run_synthesis_options* setting some of the possible options.

Example 1

This example adds the *optimize_registers* command before *compile* in the *dc_shell.synthesis_tcl* used to synthesize the test logic.

```
SETUP> set_run_synthesis_options dc_shell -pre_compilation_commands \
          optimize_registers
```

Example 2

This example adds the *-map_effort high* option to the *compile* command in the *dc_shell.synthesis_tcl* used to synthesize the test logic.

```
SETUP> set_run_synthesis_options dc_shell -compilation_options \
          {-map_effort high}
```

Related Topics

[check_synthesis](#)

[get_run_synthesis_options](#)

[report_run_synthesis_options](#)

[run_synthesis](#)

set_scan_chain_options

Context: pattern -scan

Mode: setup, when LBIST is on

Specifies options for managing scan chains within the hybrid TK/LBIST flow.

Usage

```
set_scan_chain_options {-chain_name chain_name... | -INStance instance_name
                      -Block_chain_index_list block_chain_index_list ...}
                      [-decompressor_to_scan_in_inversion {on | off} ]
                      [-scan_out_to_compactor_inversion {on | off} ]
```

Description

In LogicBIST mode, no hardware rediscovery is performed. As a result, the tool can be unaware of inversions that were inserted by a third-party tool between the scan chains and the PRPG/phase shifter on the input side and the compacter and the scan chains on the output side. Before LogicBIST fault simulation, you can specify the set_scan_chain_options command to force the inversions and proceed in such situations.

This command has no effect on ATPG pattern generation.

EDT finder can automatically rediscover the inversions between the EDT logic and scan chains, which means you can use it to export the inversions and reuse them for LogicBIST fault simulation. Use either of the following commands in analysis mode with EDT enabled (after passing K-rules) to find the inversions:

- report_scan_chains -inversions — This command reports the relevant inversions in the decompressor_inversion and compactor_inversion columns.
- write_core_description — The TCD includes information about the inversions in the decompressor_to_scan_in_inversion and scan_out_to_compactor_inversion properties within ScanChain wrappers. You can read back the TCD before performing fault simulation as described in “[Fault Simulation When There Are Inversions](#)” in the *Hybrid TK/LBIST Flow User’s Manual*.

The test patterns in –mode_internal will continue to be written out with respect to scan chain input and scan chain output. If an inversion is specified after the SCO, the unload values will not reflect the inversion. However, MISR values will take that inversion into account.

Arguments

- **-chain_name***chain_name*...

Required switch and repeatable string that specifies the name of a scan chain in the current design.

- **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list...***

Required set of switches and values that specifies a core instance and the list of chain indices starting with “1” on the EDT controller. When this is used, the masking applied to the specified chain is OX. Use this option in conjunction with the **-USed_chains** switch.

- **-decompressor_to_scan_in_inversion on | off**

Optional switch and literal pair that specifies whether the tool identifies inversions between the EDT decompressor and the SCI pin. The default is off.

- **-scan_out_to_compactor_inversion on | off**

Optional switch and literal pair that specifies whether the tool identifies inversions between the SCO pin and the EDT compactor. The default is off.

Examples

The following example makes Tesson Shell aware that there are inversions associated with both the SCI and SCO on chain3.

```
set_scan_chain_options -chain_name chain3 -decompressor_to_scan_in_inversion on  
-scan_out_to_compactor_inversion on
```

set_scan_enable

Context: dft -scan, dft -test_points

Mode: setup, analysis (dft -scan context); analysis (dft -test_points context)

Assigns scan_enable signals to specific scan chains.

Usage

```
set_scan_enable [scan_enable_pin.pathname [-Isolate]]  
[primary_input] [-Active {High | Low}]  
[-Single_global_scan_enable {ON | OFF}]  
[-CLUster_name cluster_name ... | -POwer_domain_name power_domain_name ...]  
[-Clock clock_pin]
```

Description

Assigns scan_enable signals to specific scan chains.

The set_scan_enable command can be issued in a sequence to either refine the scan enable signals assignments or overwrite the previous assignments (see the example section for this command). In general, the following rules are applied to determine how signal assignments are affected by subsequent commands:

- The most recent command that assigns a scan_enable signal takes precedence.
- If the most recent command operates on a disjoint set of scan chains, then the previous scan enable signal assignments remain intact.
- If the most recent command operates on previously specified scan chains, then previous scan_enable signal assignments are overwritten.

Arguments

- *scan_enable_pin.pathname* [-Isolate]

An optional string that specifies a pin pathname for the scan_enable signal driver. The specified pin can be either a top-level scan enable port or an internal instance pin (connection node). If an internal instance pin is specified, it must trace back to a primary input via a simple path (only inverters or buffers) or the primary_input argument.

-Isolate — Isolates new fanouts of the specified scan enable signal. Each new fanout connection is gated by an AND or NOR gate and controlled by the global test enable signal. This switch is applicable to a scan enable signal driven by a top-level port or a top-level internal instance pin only.

If no scan_enable signal driver is specified, the default scan enable name is used (scan_en).

- *primary_input*

An optional string that specifies a top-level scan_enable port. This argument supplies a primary input/top-level port for an internal instance pin specified by the scan_enable_pin.pathname argument. The specified top-level port is used when generating

the ATPG dofile and test procedure files. If the specified top-level port does not exist, it is created. The specified top-level port must be a primary input port.

- **-Active {High| Low}**

An optional switch and literal pair that specifies whether the scan_enable signal is active low or high. The default setting is high.

- **-Single_global_scan_enable {ON | OFF}**

An optional switch and literal pair that specifies to use a single scan enable signal for input wrapper chains, output wrapper chains and core chains. This is the default. The input wrapper chains scan enable signal is generated by ORing the specified or the default scan enable signal with INTEST mode test enable signal. The output wrapper chains scan enable signal is generated by ORing the specified or default scan enable signal with EXTEST mode test enable signal. If this switch is turned off, the tool uses three separate scan enable signals: scan_en_in for input wrapper chains, scan_en_out for output wrapper chains, and scan_en for core chains. Other switches are not accepted when the -single_global_scan_enable switch is used.

- **-CLUster_name *cluster_name***

An optional switch and string pair that specifies to assign a specified scan enable signal to scan chains in the specified cluster.

- **-POwer_domain_name *power_domain_name***

An optional switch and string pair that specifies to assign a specified scan enable signal to scan chains in the specified power domain.

- **-Clock *clock_pin***

An optional switch and string pair that associates the specified scan enable signal with the specified clock (clock domain). Clock_pin can be either an existing top level port (primary input pin) or an existing internal pin pathname.

An error message is issued when this switch is specified with either the -Clock Merge or the filename -Fixed option of the insert_test_logic command.

This switch can be used in conjunction with the -Edge Merge option of the insert_test_logic command.

Examples

Example 1

The following example defines two scan clusters: rx_clust and tx_clust, by assigning the cluster_name attribute to the selected instances. Each cluster has its dedicated scan enable defined with the set_scan_enable command. The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command refines the first command and makes scan chains of rx_clust controllable via the sen_rx scan enable signal. The third set_scan_enable command further refines the first command and makes scan chains of tx_clust controllable via the sen_tx scan enable signal.

```

set_scan_enable sen
set_scan_enable sen_rx -cluster rx_clust
set_scan_enable sen_tx -cluster tx_clust

set_system_mode analysis
set_attribute_value [get_scan_elements -below_instances i_rx_phy] \
  -name cluster_name -value rx_clust
set_attribute_value [get_scan_elements -below_instances i_tx_phy] \
  -name cluster_name -value tx_clust
add_scan_mode unwrapped
analyze_scan_chains
report_scan enable

```

The last command displays the following result:

```

//      command: report_scan_enable
Scan mode 'unwrapped' scan enables:
-----
Primary Input  Internal Connection Node  Scan Chain  Scan Partition
-----
/sen_rx        --                      chain0       rx_clust
-----
/sen_tx        --                      chain1       tx_clust
-----
/sen           --                      chain2       default_scan_partition
-----
```

The second set_scan_enable command operates on a set of scan chains that is entirely within the set specified in the first command; therefore, the scan chains that are in both sets will get the most recent assignment. The third set_scan_enable command operates on a set of scan chains that is disjoint from the set in the second command but is entirely contained within the first set; therefore, the scan chains that are in both the first and the third sets get the most recent assignment.

Example 2

The following example defines two scan clusters: rx_clust and tx_clust. The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command specifies an internal pin, rx_sen_reg/Q, associated with the primary input, sen_rx, as a driver of the scan enable signal controlling all scan chains of rx_clust. The third set_scan_enable command specifies an internal pin, tx_sen_reg/Q, associated with the primary input, sen_tx, as driver of the scan enable signal controlling all scan chains of tx_clust.

```

add_clock 0 clk2
set_scan_enable sen
set_scan_enable rx_sen_reg/Q sen_rx -cluster rx_clust
set_scan_enable tx_sen_reg/Q sen_tx -cluster tx_clust
set_scan_enable sen_clk2 -clock clk2

```

The last set_scan_enable command specifies that all scan chains in the clk2 clock domain are controllable by the sen_clk2 scan enable signal. The second and the third set_scan_enable commands overwrite the assignments of the first set_scan_enable command affecting chains

that are in rx_clust and tx_clust. Since the second and the third set_scan_enable commands operate on disjoint sets, they do not affect their assignments.

The fourth command will overwrite some of the assignments of the second and the third commands for scan chains that are in rx_clust and tx_clust and also in the clkInput clock domain. If you want to restrict the clock domain's assignments to a specific cluster, the -cluster and -clock options should be issued in the same set_scan_enable call.

Related Topics

[report_scan_enable](#)

[add_input_constraints](#)

[set_scan_signals](#)

set_scan_insertion_options

Context: dft -scan

Mode: setup, analysis

Specifies the high-level options for the scan insertion process.

Usage

```
set_scan_insertion_options [ -single_class_chains { on | off } ]  
[ -single_clock_domain_chains { on | off } ]  
[ -single_clock_edge_chains { on | off } ]  
[ -single_cluster_chains { on | off } ]  
[ -single_power_domain_chains { on | off } ]  
[ -single_wrapper_type_chains { on | off } ]  
[ -port_index_start_value <int> ]  
[ -port_scalar_index_modifier <int> ]  
[ -chain_length { unlimited | <int> } ]  
[ -chain_count { unlimited | <int> } ]  
[ -si_port_format <format_str> ]  
[ -so_port_format <format_str> ]  
[ -si_timing { leading_edge | trailing_edge | any_edge } ]  
[ -so_timing { leading_edge | trailing_edge | any_edge } ]  
[ -lockup_cell_type { flop | latch } ]  
[ -si_lockup_cell_type { flop | latch } ]  
[ -so_lockup_cell_type { flop | latch } ]  
[ -unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input } ]  
[ -si_pipelining { all_chains | off | wrapper_chains_only } ]  
[ -si_pipelining_type { non_scan_flop | holding_scan_flop } ]  
[ -enable_scan_cell_mapping { on | off } ]  
[ -enable_retimming { on | off | skip_synchronous_clocks } ]  
[ -port_bus_range { ascending | descending } ]
```

Description

Use this command to specify the high-level options for the scan insertion process.

Note

 Many of the options of this command are also available for the [add_scan_mode](#) and [create_scan_chain_family](#) commands. If you change any of these common options using the set_scan_insertion_options command, the tool will issue a warning that the changes will only apply when creating a new scan mode or chain family, and will have no effect on existing scan modes or chain families.

Arguments

- **-single_class_chains { on | off }**
An optional switch and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. The default is off, so the tool will allow distributing wrapper and internal cells onto the same chains.
- **-single_clock_domain_chains { on | off }**
An optional switch and literal pair that restricts a scan chain to a single clock domain. The default is off.
- **-single_clock_edge_chains { on | off }**
An optional switch and literal pair that restricts a scan chain with flops having a single edge. The default is off.
- **-single_cluster_chains { on | off }**
An optional switch and literal pair that restricts a scan chain to a single cluster using the attribute "cluster_name". The default is on.
- **-single_power_domain_chains { on | off }**
An optional switch and literal pair that restricts a scan chain to a single power domain using the attribute "power_domain_name". The default is on.
- **-single_wrapper_type_chains { on | off }**
An optional switch and literal pair that specifies that "-single_class_chains ON" controls whether to mix input wrapper and output wrapper cells in the same chain. The default is off.
- **-port_index_start_value <int>**
The starting index is 0 by default, but can be overridden by this option.
- **-port_scalar_index_modifier <int>**
An optional switch that specifies the incremental value to add to the index portion (%d) of scalar scan port names. The default is 1
- **-chain_length { unlimited | <int> }]**
An optional integer that specifies the target length of scan chains for a particular mode. By default, the no limit (unlimited) is imposed on the chain length.
- **-chain_count { unlimited | <int> }**
An optional integer that specifies the number of scan chains that should be constructed for the specified mode. By default, the chain count is set to the number of connections specified with the -si/so_connections, otherwise if the latter options were not used, then no limit (unlimited) is imposed on the chain count.
- **-si/so_port_format <format_str>**
The default si/so_port_format is set to "ts_%s_si/so[%d]" when there are more than one scan mode, otherwise it defaults to "ts_si/so[%d]". The mode placeholder "%s" gets replaced by the chain mode name, whereas the index placeholder "%d" gets substituted by

the chain index. A precision number may be used when specifying the index placeholder, for instance the format "lsi_si%2d" would cause port names like "lsi_si00", "lsi_si01", etc. to be created, note that leading zero's are inserted automatically when needed making "%2d" implicitly equivalent to "%02d".

- **-si_timing { leading_edge | trailing_edge | any_edge }**

An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in chain to control the capture edge of the first cell as leading or trailing edge. No lockup cell is inserted if "any_edge" is used as the argument. The default edge type is "leading_edge".

- **-so_timing { leading_edge | trailing_edge | any_edge }**

An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain for this scan mode. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in chain to control the change edge of the last cell as leading or trailing edge. No lockup cell is inserted if "any_edge" is used as the argument. The default edge type is "trailing_edge".

- **-lockup_cell_type { flop | latch }**

An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch.

- **-si_lockup_cell_type { flop | latch }**

An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch.

- **-so_lockup_cell_type { flop | latch }**

An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

Note

 By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

- **-unused_edt_pin_handling { auto | insert_pipelining_flop | tie_compactor_input }**

An optional switch that selects how to handle unused EDT chain pins. Options are:

insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.

tie_compactor_input — ties-off the EDT compactor.

auto — selects "insert_pipelining_flop" for EDT controllers with bypass chains, and "tie_compactor_input" for all other EDT controllers. This is the default.

- **-si_pipelining { all_chains | off | wrapper_chains_only }**

An optional switch and string that controls whether a pipelining flop gets inserted at the beginning of chains. When set to all, then all scan chains get a pipelining flop. When set to

wrapper_chains_only, then only chains of internal or external mode types and composed of wrapper scan elements get a pipelining flop. The default is wrapper_chains_only.

- **-si_pipelining_type { non_scan_flop | holding_scan_flop }**

An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

- **-enable_scan_cell_mapping { on | off }**

An optional switch and literal pair that controls the mapping from non-scan to scan cells. The default is on.

- **-enable_retiming { on | off | skip_synchronous_clocks }**

An optional string and literal pair that provides the ability to control the insertion of lockup (re-timing) cells between the scan chain elements. skip_synchronous_clocks omits the insertion of retiming between synchronized clock domains. Use the [add_synchronous_clock_group](#) command to define the synchronous clocks. The default edge type is on.

- **-port_bus_range { ascending | descending }**

An optional switch and literal pair that determines whether created bus ports are created with ascending range (for example, [0:31]) or descending range (for example,[31:0]). The default is descending.

Examples

The following example demonstrates using the set_scan_insertion_options command to disable restricting a scan chain to a single power domain using the attribute “power_domain_name” and set the type of lockup (re-timing) cell between the scan chain output pin and the last scan cell to a flop:

```
set_scan_insertion_options -single_power_domain_chains off -lockup_cell_type flop
```

set_scan_signals

Context: dft -scan, dft -test_points

Mode: setup

Specifies the input control signals' drivers used by the [insert_test_logic](#) command.

Usage

```
set_scan_signals [-TEn pathname] [-int_test_enable pathname][-ext_test_enable pathname]
                 [-TClk pathname] [-SET pathname] [-RESet pathname]
                 [-READ pathname] [-Write pathname]
                 [-Muxed | -Disabled] [-Active {High | Low}]
```

Description

Specifies the input control signals' drivers used by the [insert_test_logic](#) command.

Additional control pins are specified with [set_scan_enable](#), [set_bidi_gating](#), and [set_tristate_gating](#) commands.

The tool uses the write control and read control pins when inserting test logic for RAMs. DRC verifies whether the RAMs can be held off when the write control pin is off. No test logic is inserted for RAMs that can be held off.

If the new scan design is intended for Tessent FastScan/Tessent TestKompress, and the write clock input of the RAM is not controllable, it is multiplexed out by a new write clock. The selector to the mux is the test enable pin.

You can use instance pin pathnames to define the test enable and clock pin names. The instance pathnames must trace back through a simple path (inverters or buffers only) to a primary input/output of the design, or you must manually develop the test procedure file and the dofile before running ATPG.

Arguments

- **-TEn pathname**

An optional switch and string pair that specifies a name for the test enable pin. By default, the test enable pin is named `test_en`.

Use the `-Active` switch to specify the active state of the Test enable signal.

- **-int_test_enable pathname**

An optional switch and string pair that specifies a name for the test enable pin in the INTEST mode. By default, the test enable pin is named `int_ltest_en`.

- **-ext_test_enable pathname**

An optional switch and string pair that specifies a name for the test enable pin in the EXTEST mode. By default, the test enable pin is named `ext_ltest_en`.

- **-TClk *pathname***

An optional switch and string pair that specifies a pathname for the test clock pin for all scan types. If no test clock pin exists, one is created using the specified name. By default, the new test clock pin is named `test_clk`.

If no test clock pin is specified, an existing clock pin from the design is used. Eligible clock pins must be defined with the `add_clocks` command and are capable of capturing data in a sequential cell without test logic in the clock path. The clock pin is selected at random. If no clock pins are found, a new one is created and named `test_clk`.

If an internal pin is specified as the test clock pin, it is traced back to a predefined scan clock. If a predefined scan clock is found, the off-state value is used for test logic insertion. This tracing can only be performed along a single path that consists of only straight wire, inverters, and buffers.

- **-SET *pathname***

An optional switch and string pair that specifies a pathname for the set pin used for the flip-flop or latch when inserting test logic. By default, the set pin is named `scan_set`.

Use the `-Active` switch to specify the active state of the Set signal.

- **-RESet *pathname***

An optional switch and string pair that specifies a pathname for the reset pin used for the flip-flop or latch when inserting test logic. By default, the set pin is named `scan_reset`.

Use the `-Active` switch to specify the active state of the Reset signal.

- **-READ *pathname***

An optional switch and string pair that specifies a pathname for the read control pin used for RAM when inserting test logic. By default, the read control pin is named `read_clk`.

- **-WRITE *pathname***

An optional switch and string pair that specifies a pathname for the write control pin used for RAM when inserting test logic. By default, the write control pin is named `write_clk`.

- **-Muxed | -Disabled**

An optional switch that determines how the specified set, reset, write control, or read control lines are gated. These options can only be used with `-Set`, `-Reset`, `-Write`, and `-Read` switches specified in the same command.

Options include:

-Muxed — Multiplexes all set and reset pins with the original signal. Only set and reset pins defined as clocks are affected. Default setting.

-Disabled — Uses an AND gate with the test enable signal to disable the set and reset inputs of flip-flops and the SEN type scan enable signal to disable the write and read clocks.

- -Active High | Low

An optional switch and literal pair that specifies the active state of the Test enable, Set or Reset signals. Within a single command, you can only use the -Active switch once.

The -Active switch applies to any of the above signals that are specified in the command with the -Ten, -Set, and -Reset switches. By default, all these signals are active on high. The -Active switch is not applicable to any of the other signals specified in this command.

Examples

Example 1

The following example specifies an internal pin, **modxyz/nand02/test_en_L**, as the driver of the Test enable signal to be used during the scan insertion.

```
add_clocks 0 clock
set_scan_signals -ten modxyz/nand02/test_en_L -active low
set_system_mode analysis
insert_test_logic
```

Example 2

The following example shows how to set different controls with successive `set_scan_signals` commands.

```
set_scan_signals -ten TEST_MODE
set_scan_signals -reset RESET_L -active low
set_scan_signals -write ATPG_WRT_INH -read ATPG_READ_INH
set_scan_signals -set ATPG_SET -disabled
```

Related Topics

[insert_test_logic](#)
[set_bidi_gating](#)
[set_scan_enable](#)
[set_tristate_gating](#)

set_screen_display

Context: unspecified, all contexts

Mode: all modes

Specifies whether the tool writes the transcript to the session window.

Usage

`set_screen_display ON | OFF`

Description

Specifies whether the tool writes the transcript to the session window.

If you create a logfile with the `set_logfile_handling` command, you may want to disable the tool from writing the same information to the session transcript window.

Arguments

- **ON**
A literal that enables the tool to write the session information to the transcript window. This is the default behavior upon invocation of the tool.
- **OFF**
A literal that disables the tool from writing any of the session information to the transcript window, including error messages.

Examples

The following example shows how to use the logfile functionality to capture the transcript in a file and then disable the tool from writing the transcript to the display.

```
set_logfile_handling /user/design/setup_file  
set_screen_display off
```

Related Topics

[set_logfile_handling](#)

set_shadow_check

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -scan context only)

Specifies whether the tool identifies sequential elements as “shadow” elements during scan chain tracing.

Usage

```
set_shadow_check {ON | OFF | REstricted} [-ALLOW_cross_clock_domains {ON | OFF}]
```

Description

Specifies whether the tool identifies sequential elements as “shadow” elements during scan chain tracing.

You can use the set_shadow_check command to disable the checking and avoid corresponding error messages. This prevents identification of any non-scan sequential element as a shadow element.

Arguments

- **ON**
A literal that enables shadow checking. This is the initial state upon invocation of the tool.
- **OFF**
A literal that disables shadow checking.
- **REstricted**
A literal that disables the tool’s identification of shadow cells that require independent shifts in certain pattern formats (for example, parallel Verilog format).

Note

 An independent shift results from a one-shift “apply shift” statement in the load_unload procedure. For more information, refer to the description of the [P19](#) rule.

- **-ALLOW_cross_clock_domains {**ON** | **OFF**}**

An optional switch and literal pair that enables consideration of cross-clock domain state elements as shadow cells. This option can increase test coverage, but you should turn on this switch only when you are sure that the clocks are balanced and the shift values to the cross-domain cells are reliable.

Examples

The following example disables shadow checking:

```
set_shadow_check off
```

Related Topics

[report_environment](#)
[set_drc_handling](#)
[write_patterns](#)

set_shift_register_identification

Context: dft -scan, dft -test_points

Mode: setup

Enables/disables shift register identification. The default is ON.

Usage

```
set_shift_register_identification {ON | OFF}  
[-ALLOW_COMBINATIONAL_logic_between_registers {ON | OFF}]
```

Description

Enables/disables shift register identification. The default is ON.

For more information about automatic identification of shift registers, see “[Automatic Recognition of Existing Shift Registers](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **ON**

A required literal that enables shift registration identification. This is the default.

- **OFF**

A required literal that disables shift registration identification.

- **ALLOW_COMBINATIONAL_logic_between_registers {ON | OFF}**

An optional switch and literal pair that specifies for the tool to consider or ignore the use of the DRC calculated state-stability values during shift register identification. When the switch is Off, the constraint values that sensitize a data path between sequential cells are ignored. As a result, multiple-input gates (AND, OR, MUX, etc.) between sequential cells cannot be traced during shift register identification. Single-input gates (BUF and INV), on the other hand, can always be traced. When the switch is On, the constraint values that sensitize a data path between sequential cells are considered during shift register identification tracing. Multiple-input gates do not block the tracing as long as a single path is sensitized through them.

OFF

A literal that specifies ignoring the constraint values. This is the default.

ON

A literal that specifies considering the constraint values.

Examples

The following example disables automatic shift register identification.

```
set_shift_register_identification off
```

Related Topics

[report_scan_cells](#)

[report_shift_registers](#)

set_silicon_insight_option

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Specifies optional settings for running Tessent SiliconInsight.

Usage

```
set_silicon_insight_option [ -cdp_verification { on | off } ]  
[ -ignore_uncontacted_pin_data { on | off } ]  
[ -interactive_ijtag { on | off } ]  
[ -interactive_ijtag_pattern_type { SVF | STIL } ]  
[ -maximum_failing_cycles number ] [ -simdut_port name ]  
[ -simdut_timeout integer ]  
[ -maximum_failing_cycles max_failing_cycles ]
```

Description

The `set_silicon_insight_option` command specifies optional global settings that determine the behavior of subsequent Tessent SiliconInsight sessions. These global settings remain in effect until you change them.

Some of the options apply to particular Tessent SiliconInsight modes of operation, such as SimDUT mode, and others are general.

Arguments

- `-cdp_verification { on | off }`

An optional switch that specifies whether to automatically perform CDP verification. Set this option to “off” when you are not using a TSDB. For example, when you are running custom tests added using the `add_cdp_test` command.

In addition, this switch ensures that new test patterns added as part of a test were generated by Mentor tools (hence will contain a signature that links it to the design it was generated for). You may want to turn verification off if you are adding test patterns that were not generated by Mentor tools.

- `-ignore_uncontacted_pin_data { on | off }`

An optional switch that specifies whether to ignore uncontacted pin data on the selected_setup adaptor. You may want to set this option to “on” when, for example, you are running a custom test that uses more pins than are declared contacted in the pinmap and you know that these extra pins can safely be ignored.

- `-interactive_ijtag { on | off }`

An optional switch that specifies whether to turn on interactive IJTAG so that you can debug a device at the PDL level. See section “[PDL-Level Debugging for Tessent IJTAG-Inserted Devices](#)” in the *Tessent SiliconInsight User’s Manual for Tessent Shell* for details.

- **-maximum_failing_cycles *max_failing_cycles***

An optional switch that enables you to control the maximum number to capture failing cycles in pattern execution. By default, the tool collects and processes all failures. In cases of potential catastrophic defects, pattern execution can take a long time and result in unusable data. This option allows you to collect and process subsets of the failures.

- **-interactive_ijtag_pattern_type { SVF | STIL}**

An optional switch and string pair that specifies whether to generate SVF or STIL patterns for interactive IJTAG analysis and debugging. The default is SVF.

- **-maximum_failing_cycles *number***

An optional switch and integer pair that specifies the maximum number of raw failure cycles to collect during ATPG pattern execution. Use this option when you want to execute a subset of patterns within the STIL pattern file. The tool executes the full pattern but only collect the failing cycles for the specified subset. The default is 2000000000.

- **-simdut_port *name***

An optional switch and string pair that specifies the name of the SimDUT port.

- **-simdut_timeout *integer***

An optional switch and integer pair that controls the length of time before SimDUT mode times out. You can set this option to ensure that the tool does not time out before elaboration completes. Specify the time in seconds. The default is 600.

Examples

The following example turns on interactive IJTAG debugging.

```
set_silicon_insight_option -interactive_ijtag on
```

set_simulation_library_sources

Context: unspecified, all contexts

Modes: all modes

Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the [run_testbench_simulations](#) command.

Usage

```
set_simulation_library_sources {[-v file_path ...] [-y dir_path ... -extensions ext ...]}  
  [-sv_extensions ext ...]  
  [-logical_library_map_list name path2dir ...] }  
  [-include_directory_list dir_path ...]  
  [-f file_path ...]  
  | [-clear]
```

Description

Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the [run_testbench_simulations](#) command. The *-logical_library_map_list* is used to reference already compiled libraries.

When you use the *-logical_library_map_list* option, you must make sure to reference pre-compiled libraries that are consistent with the simulator and simulation version you are using with the [run_testbench_simulations](#) command. For example, a pre-compiled library compiled with the Questa 7.2 vlog command cannot be simulated with a different simulator nor with a newer version of vsim.

All switches for this command are interrelated. Each switch is set to their new value each time this command is issued. For example, specifying the command without the *-v* switch does not preserve the previously set *-v* values but instead clears them.

Arguments

- *-v file_path ...*
An optional repeatable switch and value pair that specifies one or more files in which to search for Verilog simulation modules. The relative order of all *-v* and *-y* switches are preserved: the library modules are searched from the first one to the last, and the *-v* files are searched from top to bottom.
- *-y dir_path ...*
An optional repeatable switch and value pair that specifies one or more directories in which to search for files matching Verilog simulation module names. Files that have a leaf name that matches a module name and have one of the extensions specified with the *-extension* switch are automatically loaded if the module does not already exist in memory. You must specify the *-extension* option when you have one or more usages of the *-y* switch. The

relative order of all -v and -y switches is preserved, and the library modules are searched from the first one to the last.

- **-extensions ext**

An optional non-repeatable switch and value pair that specifies the list of extensions that are used to search the specified -y directories for files matching the module names. You must specify the -extension switch once and only once when you specify the -y switch one or more times.

- **-sv_extensions ext**

An optional repeatable switch and value pair that specifies the list of SV extensions that are used to search the specified directories matching the module names.

- **-logical_library_map_list name path2dir**

An optional non-repeatable switch and value pair that specifies a list of logical library name and directory name pairs. Each of those pairs associates a physical directory to a pre-compiled logical library name. You can map multiple logical libraries to their respective directories by specifying multiple pairs of elements. The pre-compiled libraries must be consistent with the simulator and simulator version that you are using in the [run_testbench_simulations](#) command. For example, a pre-compiled library compiled with the Questa 7.2 vlog command cannot be simulated with a different simulator nor with a newer version of vsim.

- **-include_directory_list dir_path**

An optional switch and repeatable string that specifies one or more directory paths that contain `include files.

- **-f file_path ...**

An optional switch and Tcl list that specifies the name(s) of one or more ASCII files that contains pointers to design files. The referenced file is usually generated by another tool, so the -f option allows you to conveniently use the file rather than to reformat the contents of the file into a set_simulation_library_sources command line. The file can contain only the following elements, one per line:

- *design_path* — A string that specifies the name of a Verilog file to load.
- *-v design_path* — A string that specifies the name of a Verilog file to load.
- *-y dir_path* — A string that specifies a path to a directory that contains Verilog files to load.
- *+libext+file_extension* — A string that specifies which files in the “-y dir_path” to load. For example: “+libext+.v” loads all .v files.
- *+define+XXX[=YYY]* — A string that defines a macro called “XXX” with an optional value of “YYY”. The defined macro is used when compiling any files listed in the -f file.

- `+incdir+dir_path1+dir_path2+...` — A string that defines one or more directory paths to directories that contain files that use the ‘include directive.

If you use the `-f` switch with the `set_simulation_library_sources` command, the `file_path` can be relative; however, when specifying the `-f` switch with the [run_testbench_simulations](#) command, you must specify an absolute path for `file_path`. The `file_path` must be an absolute path to the file it references because the “compilation options” is used verbatim and not interpreted at all by the `run_testbench_simulation` command as follows:

```
run_testbench_simulations -compilation_options { -f file_path }
```

In all cases, the content of the file referenced by `file_path` must include only absolute paths to files/directories.

- `-clear`

An optional switch that clears the previously specified values for all other switches in this command as if they had never been specified before. This switch is mutually exclusive with all other switches.

Examples

The following example defines the simulation library sources to be a directory called “./mems” which contains the memory simulation files with a .v extension and a directory named “./tsmc130_questa_10.2” in which the TMSC Verilog simulation cells were pre-compiled using vlog version 10.2. The assumption is that the [run_testbench_simulations](#) command uses Questa as its `-simulator` value and that the Unix search path environment is such that it will pick the Questa commands associated with version 10.2.

```
> set_simulation_library_sources -y ./mems -extensions v \
    -logical_library_map_list tsmc130 ./tsmc130_questa_10.2
> run_testbench_simulation -simulator questa
```

Related Topics

[report_simulation_library_sources](#)

[run_testbench_simulations](#)

[set_simulation_options](#)

Context: all contexts

Mode: setup, analysis, insertion

The `set_simulation_options` command controls the behavior of the tool during pattern generation and simulation.

Usage

```
set_simulation_options [-SET_Reset_dominant_port {OFF|ON}]
[-C6_mask_races {OFF|ON}]
[-MULTIcycle_fault_simulation {ON|OFF}]
[-MUX_Select_x_sim_x {OFF|ON}]
[-FEEDback_buffer_x_init_value {ON|OFF}]
[-REPORT_X_capturing_cells {OFF|ON}]
[-REPORT_Race_masking {OFF|ON|VERbose}]
[-SIMULATE_LEARNED_SEQuential_tie_gates {OFF|ON}]
```

Arguments

- [-SET_Reset_dominant_port OFF|ON](#)

An optional switch and literal pair that enables or disables setting the Set/Reset port as the dominate synchronous port for both flip-flops and latches when multiple ports are on simultaneously. When the switch is on, and either Set or Reset port is on (but not both on), the evaluation value of the state element is dependent on the Set or Reset port. When the switch is off and multiple ports are on simultaneously, the evaluation value is dependent on all enabled ports. When enabled ports agree on the same value, the state element is set to the agreed value. Otherwise, the state element is set to X. Also, the state element is set to X when Set and Reset port are both on simultaneously despite this switch setting.

[OFF](#)

A literal that specifies disabling set/reset as the dominate port.

[ON](#)

A literal that specifies enabling set/reset as the dominate port. This is the default.

- [-C6_mask_races OFF|ON](#)

An optional switch to turn on pessimistic simulation on C6 violated state elements and mask the capture values only when the data port transitions at the same frame the clock port captures. This option allows you to change the handling of C6 violated cells when data and clock ports have a race transition. By default, `create_patterns` will turn on `clock_off` simulation and the tool assumes that the clock ports transition is earlier than the data ports. If this assumption is not true, this option is recommended to mask the C6 violated cells to prevent mismatches due to races. This functionality is less pessimistic than the command “`add_cell_constraints -DRC C6`” as the latter command will mask C6 violated cells even when no races happen.

OFF

A literal that disables this functionality. This is the default.

ON

A literal that enables this functionality.

- **-MULTIcycle_fault_simulation ON|OFF**

An optional switch and literal pair that determines whether or not the tool runs multicycle path ATPG and fault simulation. The default is ON. When this switch is ON, ATPG creates patterns to detect faults on the multicycle paths and fault simulation allows the fault delay on the multicycle path to be $(n+1)$ cycles, where n is the cycle multiplier of the multicycle path. When a fault pin is included in multiple multicycle path cones, the fault delay is determined by the smallest cycle multiplier.

- **-MUX_Select_x_sim_x OFF|ON**

An optional switch and literal pair that specifies to model _mux primitives in the design as either consensus or non-consensus. By default, the tool always models _mux as consensus (evaluates _mux with a known value as long as both data ports are the same known value).

You can use the ON argument to model the _mux primitive as non-consensus (output X for every _mux instance whose select line is X). Note that changing _mux primitives to non-consensus can reduce the test coverage. Also, you can use this switch in any mode, but when you use it in setup mode, the tool evaluates _mux instances as non-consensus mux instances, which can result in additional DRC violations.

- **-FEEDback_buffer_x_init_value ON|OFF**

An optional switch and literal pair that allows the FB_BUF primitive to carry over its value from the previous cycle instead of being initialized to X. The default is ON, which means that the FB_BUF is still initialized with X value at the beginning of a simulation cycle. You can choose to turn off X initialization to get less pessimistic simulation results. Note that turning off this option increases the chance of pattern simulation mismatches if all FB_BUF primitives in the design cannot be safely simulated in this manner.

- **-REPORT_X_capturing_cells {OFF|ON}**

An optional switch and literal pair that reports scan cells that capture X during simulation. When you set this option to ON, the tool displays this message when detecting a parallel pattern that captures X:

```
// 10 scan cells capture X
```

You use “[report_patterns -unload_x](#)” to find the patterns that capture X on scan cells. In addition, when you specify the -pattern_index switch with -unload_x, the report_patterns command reports the scan cells that capture X on the specified pattern.

- **-REPORT_Race_masking {OFF|ON|VERbose}**

An optional switch and literal pair that enables the tool to report when races or glitches are detected that cause sequential element capture values to be masked. These are reasons for races or glitches that can cause the sequential elements to be masked:

- C6 races — The sequential element is masked due to C6 port races, where the C6 data port value is changing in the same frame as the clock port capture frame.
- Glitches on set/reset port in the scan cell loading frame — The set or reset port has potential glitches and the sequential value can be different depending on the glitch value.
- Set or reset port race with the clock port — The set or reset port is de-asserted in the same frame as the clock port capture frame and the capture value can be different if the de-assert happens before or after the clock port capture event.

This message is reported during the good machine simulation when this switch is set to “on”:

```
// Warning: Forcing gate '/cm0507/cm2304/' (7956) to X due to
// glitch at set or reset port for pattern 7.
```

Note

 To prevent excessive messages, only the first ten masked cell are reported for by this tool for race and glitch masking cases (except C6 races). If there are more than ten masked cells, this message is reported:

```
// Only the first 10 have been reported.
// Use 'report_gates -glitch_masked_cells' to report all masked
// cells.
```

- **-SIMULATE_LEARNED_SEQuential_tie_gates {OFF|ON}**

An optional switch and literal pair used to change the ATPG simulator behavior for sequential gates learned as tied during DRC. When you specify off, the switch default, the tool will maintain such sequential tie gates at the learned tie values during simulation.

When you specify on, the tool will not use the learned tie values and evaluate all such sequential gates during simulation.

One possible use for this switch would be to resolve Verilog simulation differences due to X handling on a control port, where the Verilog simulator pessimistically evaluates the gate as X, even though a constant value can be preserved in silicon.

Examples

Example 1

The following example demonstrates disabling setting set/reset as the dominate synchronous port:

```
set_simulation_options -set_reset_dominant_port off
```

Example 2

The following example illustrates how to change the tool to model all _mux instances as non-consensus for both DRC checking and ATPG:

```
SETUP> set_simulation_options -mux_select_x_sim_x on
SETUP> set_system_mode analysis
// Flattening process completed, cell instances=28234, gates=45400,
// PIs=110, POs=103, CPU time=93.00 sec.
// -----
// Begin circuit learning analyses.
// -----
// Equivalent gates=54    classes=0    CPU time=9.00 sec.
...
ATPG> create_patterns
```

Related Topics

[add_cell_constraints](#)
[get_simulation_option](#)
[report_patterns](#)
[set_clock_off_simulation](#)

set_skewed_load

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool includes a skewed load in the patterns.

Usage

`set_skewed_load OFF | ON`

Description

Specifies whether the tool includes a skewed load in the patterns.

The `set_skewed_load` command either allows patterns to include a skewed load or restricts patterns from including a skewed load. You can use this command in any system mode, but if you use it outside the Setup mode, you must provide a `skew_load` procedure in the test procedure file. If the `skew_load` procedure does not exist, Tesson FastScan issues an error.

You use the `skew_load` procedure in LSSD designs to place different data in the master and slave of a scan cell by applying an additional clock pulse to the master shift clock.

Arguments

- **OFF**

A literal that specifies to not include a skewed load in the patterns. This is the default behavior upon invocation of Tesson FastScan.

- **ON**

A literal that specifies to include a skewed load in the patterns.

Examples

The following example specifies for patterns to include the skewed load:

```
set_skewed_load on
set_system_mode analysis
create_patterns
```

set_split_capture_cycle

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Controls whether the tool updates simulation data between clock edges.

Usage

set_split_capture_cycle ON | OFF

Description

Controls whether the tool updates simulation data between clock edges.

Tip

i If you use ATPG Expert (via the [create_patterns](#) command), you do not need to issue this command. ATPG Expert will automatically enable this command when it is needed. You may get the best results by letting ATPG Expert decide when to use this command.

The `set_split_capture_cycle` command enables or disables the simulation of level-sensitive and leading edge state elements updating as a result of applied clocks. When set to on, the tool updates simulation data between clock edges. This “split capture cycling” of data allows the tool to determine correct capture values for trailing edge and level-sensitive state elements, even in the presence of C3 and C4 violations. When set to “off”, simulation data is updated once each clock cycle.

Note that the `set_split_capture_cycle` command affects ATPG (subsequent [create_patterns](#) commands) but not logic simulation (subsequent [simulate_patterns](#) commands).

Arguments

- **ON**
A literal that specifies to update ATPG data between clock edges. This is referred to as split-capture cycling.
- **OFF**
A literal that specifies to not update ATPG data between clock edges. This is the default behavior upon invocation of the tool.

Examples

The following example enables split capture cycling:

set_split_capture_cycle on

Related Topics

[set_clock_off_simulation](#)

[set_pattern_type](#)

set_stability_check

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.

Usage

For dft -scan or when EDT is Off

```
set_stability_check {ON | OFF | All_shift}
    [-SIM_Static_atpg_constraints {OFF|ON}]
    [-MAX_shift_cycles {max_number | DEfault | LOngest_chain_length}]
    [-SIM_Capture_procedures
        {ON | Off | ON_Exclude_empty_capture_cycle_for_chain_test}]
```

When EDT is On

```
set_stability_check {ON | OFF}
    [-SIM_Static_atpg_constraints {OFF|ON}]
    [-MAX_shift_cycles {max_number | DEfault | LOngest_chain_length}]
    [-SIM_Capture_procedures
        {ON | Off | ON_Exclude_empty_capture_cycle_for_chain_test}]
```

Description

Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.

In order to perform scan chain tracing, design rules checking (DRC) sometimes requires values to be present from non-scan state elements. For example, a control register in a test controller, such as a JTAG TAP, needs to be checked to see that the TAP register holds state during the shift procedure. By default, this checking is performed. This command controls the level of checking.

For detailed information about the use and reporting of state stability, refer to the “[State Stability Issues](#)” appendix in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- **ON**

A literal that enables the tool to perform a fast check of the effect of applying the main shift procedure on non-scan cells. This is the default behavior upon invocation of the tool.

- **Off**

A literal that disables the tool from performing any checks on the effect of applying the main shift procedure on non-scan cells.

- **All_shift**

A literal that enables the tool to perform the most detailed level of checking. The main shift procedure is simulated for as many applications as the procedures call for. When you specify this option, it can significantly increase your run time.

Note

 This option is primarily for logic BIST applications where the contents of shift registers need to be known at the end of scan loading, at the beginning of the capture cycle.

- **-SIM_Static_atpg_constraints OFF | ON**

An optional switch and literal pair that enables or disables the simulation of static ATPG constraints during DRC's state stability analysis. Static ATPG constraints are those you specify using the -Static switch with the add_atpg_constraints command.

When you enable simulation of static ATPG constraints, the tool will assume the constraints are satisfied and will simulate their values during DRC's state stability analysis of nonscan memory elements. This simulation is useful in some clock gating situations.

The invocation default is for DRC not to perform this simulation. The argument choices are as follows:

OFF — A literal that disables the simulation of static ATPG constraints during DRC.
This is the invocation default.

ON — A literal that enables the simulation of static ATPG constraints during DRC.

- **-MAX_shift_cycles max_number | DEfault | LOngest_chain_length**

An optional switch and integer or literal pair that limits the number of shift cycles. You can only use this switch when stability check is set to ON (set_stability_check ON).

- The *max_number* is an integer greater than or equal to 1 (one) that limits the maximum number of shift cycles the state stability analysis can simulate. When setting *max_number*, you should follow the guidance of the **D12** DRC warning message for the number of shift cycles; the tool automatically considers existing independent shift cycles when calculating this recommended number.
- Use the **DEfault** option with this switch to revert to the default behavior of state stability analysis without imposing a specific shift cycle limit using the following syntax:

```
set_stability_check ON -MAX_shift_cycles default
```

- The **LOngest_chain_length** option instructs the tool to use the full-chain length as the iteration limit instead of the default. Using this option can significantly increase runtime.

- **-SIM_Capture_procedures {ON | OFF| ON_Exclude_empty_capture_cycle_for_chain_test}**
(setup mode only)

An optional switch and literal pair that enables a more accurate stability analysis for the NCP (named capture procedure) flow. The two “ON” options are useful when all capture sequences are using predefined NCPs. These options are needed only if the events applied in the NCP matter to state stability and the tool's ability to pass DRC. When performing state stability in preparation for DRC, the tool usually simulates the clocks as *off_value-X-off_value*. However, if the exact capture sequence to be applied during capture needs to be simulated accurately so the state of a clock or shift controller is in the right state, then you can use this option for that purpose.

By default, the more accurate stability analysis for NCPs is turned OFF. The argument choices are as follows:

ON — A literal that enables a more accurate stability analysis for the NCP flow. During state stability analysis for capture windows, the tool simulates every NCP, plus a virtual NCP that includes an empty cycle that does nothing (in order to mimic a chain test).

For each gate, the tool takes the common simulation values at the end of all NCPs (and includes the empty cycle NCP) as the end value of capture window. That is, the gate's final stability value at the end of capture is 1 or 0 if and only if its simulation value at the end of all NCPs is 1 or 0; otherwise, the final stability value is X. This is still pessimistic in the sense that only the common values of all NCPs are preserved, but it is more accurate because the simulation of the capture is exactly as specified by those NCPs.

For the case of an NCP with multiple loads, the tool splits the NCP during simulation, so there is an NCP for each individual load.

OFF — A literal that disables the more accurate stability analysis for the NCP flow. This is the default.

ON_Exclude_empty_capture_cycle_for_chain_test — A literal that enables the more accurate stability analysis, but specifies that the tool not include the virtual NCP with an empty cycle in the stability analysis (as with the ON option). Also, the tool does not save chain test patterns if the stable values at the beginning and end of the capture window for all NCPs are not the same.

Note that once the tool has performed DRC stability analysis based on the NCPs, you must only run subsequent ATPG operations using those NCPs.

For more information about the effect of using this switch on gate reporting, refer to “[Example 12 — Basic with Enhanced Stability Check for NCPs](#)” in the *Tessent Scan and ATPG User's Manual*.

Examples

Example 1

The following example enables fast stability checking of the effect of applying the main shift procedure on non-scan cells:

```
set_stability_check on
```

Example 2

The following example enables fast stability checking as well as the simulation of static ATPG constraints during DRC. Notice the use of the report_environment command to confirm the resultant changes in tool settings:

```
report_environment
```

```
...
split capture cycle =      OFF
stability check =         OFF
...
```

```
set_stability_check on -sim_static_atpg_constraints on
report_environment
```

```
...
split capture cycle =      ON
stability check =          ON
...
```

Related Topics

[set_gate_report](#)

set_static_dft_signal_values

Context: dft patterns

Mode: setup

Prerequisites: ICL is elaborated.

Sets the static DFT signal that exists in the design such that the signal can be used as part of a test_setup.

Usage

```
set_static_dft_signal_values
[-all | -current_design | -icl_instances icl_instances | -instances instances]
dft_signal_value_pairs
```

Description

Sets the static DFT signals that exist in the design such that the signals can be used as part of a test_setup. All static DFT signals implemented with IJTAG can be set using this command. See the [add_dft_signals](#) command for information on adding the DFT signals. The DFT signals are grouped by each module in which you added the DFT signals and ran the [process_dft_specification](#) command. These modules have the tesson_design_level attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value from the following using the heuristic described below. An implicit value will result in an iWrite command in the iCall when the implicit value is different than the reset value. Explicit value always results in an iWrite even if the value matches the reset value of the signal. Use the [report_static_dft_signal_settings](#) command to see the effect of the set command. A - in the value column means the value will be kept to its reset value:

- When setting a DFT signal of usage “scan_mode”:
 - All other DFT signals within the same instance with usage scan_mode are set to 0 unless the signal had a previous explicit setting.
 - All DFT signals within the same instance with usage “logic_test_control” are set to their “value_in_pre_scan_drc” values unless the signal had a previous explicit setting. If the “value_in_pre_scan_drc” value is an X, it is converted to a 0.
 - If the scan_mode_type is “internal”, the int_ltest_en DFT signal within the same instance is set to 1 unless the signal had a previous explicit setting.
 - If the scan_mode_type is “external”, the ext_ltest_en DFT signal within the same instance is set to 1 unless the signal had a previous explicit setting.
 - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.

- When setting a DFT signal of usage “logic_test_control”:
 - All DFT signals within the same instance with usage “logic_test_control” are set to their “value_in_pre_scan_drc” values unless the signal had a previous explicit setting. If the “value_in_pre_scan_drc” value is an X, it is converted to a 0.
 - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.
- When setting a DFT signal of usage “global_dft_control”:
 - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.

To remove a previous explicit setting, use the [reset_static_dft_signal_values](#) command.

Arguments

- **-all**

An optional switch that applies the setting to the specified DFT signals found within all ICL instances having the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also set.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-current_design**

An optional switch that applies the setting to the specified DFT signals found within top module. The top-level module must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-icl_instances *icl_instances***

An optional switch that applies the setting to the specified DFT signals found within the specified ICL instances. The specified ICL instances must have the `tessent_design_level` attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The `-all`, `-current_design`, `-icl_instances`, and `-instances` switches are mutually exclusive. When unspecified, the `-current_design` switch is the default.

- **-instances *instances***

An optional switch that applies the setting to the specified DFT signals found within the specified design instances. The specified design instances must match the value of the “`hierarchical_design_instance`” attribute found on an ICL instance with the

tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- *dft_signal_value_pairs*

A repeatable set of string pairs used to specify DFT signal name and value pairs. The string list must include an even amount of elements where the odd elements correspond to DFT signal names and the even values correspond to 0 or 1. The string can also be a single well-formatted Tcl list containing an even number of element where the odd elements correspond to DFT signal names and the even values corresponds to 0 or 1.

Examples

The following example shows the use of the `set_static_dft_signal_values` followed by `report_static_dft_signal_settings` and the `get_static_dft_signal_icall` inside an `open_pattern_set`.

Notice how the DFT signals start with no explicit nor implicit setting. You can use the `reset_static_dft_signal_values` to return to this state after you have specified the settings. In the example, the “int_edt” DFT signal is set to 1. Consistent with the algorithm described above, the other DFT signals of usage “scan_mode” are left to stay in their reset state. The DFT signals of usage “logic_test_control” are set to their “value_in_pre_scan_drc” values. Because “int_edt” is a scan_mode of type “internal”, the int_ltest_en is also asserted high. The DFT signals of usage “global_dft_control” are automatically set to the “default_value_in_all_test” values. Remember that implicit values matching the reset values are shown as -. No iWrite commands are issued for them as their reset states are already the needed values.

To reuse this scan mode in an unwrapped manner, you explicitly set int_ltest_en to 0 to prevent the signal from being implicitly set. Similarly, you can explicitly set DFT signals of usage “global_dft_control” to a value that is different than their “default_value_in_all_test” values.

As described in the `get_static_dft_signal_icall` command description section, the `get_static_dft_signal_icall` command is automatically invoked to configure the test_setup when in a context that supports `set_test_setup_icall`.

```

> set_context patterns -ijtag
> read_icl ../../sub_block_13.icl
> set_current_design sub_block_13
>
> set_system_mode analysis
> open_pattern_set P1
> report_static_dft_signal_settings

// ICL Module      : sub_block_13

// -----
// DFT Signal Name Usage          Set value Set source
// -----
// all_test          global_dft_control   -      -
// ltest_en          logic_test_control   -      -
// int_ltest_en     logic_test_control   -      -
// ext_ltest_en     logic_test_control   -      -
// int_edt          scan_mode(internal)  -      -
// ext_multi         scan_mode(external) -      -

//
> set_static_dft_signal_values int_edt 1
> report_static_dft_signal_settings

// ICL Module      : sub_block_13

// -----
// DFT Signal Name Usage          Set value Set source
// -----
// all_test          global_dft_control   1      Inferred
// ltest_en          logic_test_control   1      Inferred
// int_ltest_en     logic_test_control   1      Inferred
// ext_ltest_en     logic_test_control   0      Inferred
// int_edt          scan_mode(internal)  1      Explicit
// ext_multi         scan_mode(external) 0      Inferred

//
> puts [get_static_dft_signal_icall]
dft_signal_iproc
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.all_test 1
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.int_edt 1
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.ext_muti 0
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.int_ltest_en 1
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.ext_ltest_en 0
    sub_block_13_rtl2_tessent_tdr_sri_ctrl_inst.ltest_en 1
> iCall {*} [get_static_dft_signal_icall]
> close_pattern_set
> write_patterns p1.v -verbose

```

Related Topics

[add_dft_signals](#)
[get_static_dft_signal_icall](#)
[report_static_dft_signal_settings](#)
[reset_static_dft_signal_values](#)

set_static_learning

Context: unspecified, all contexts

Mode: setup

Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.

Usage

```
set_static_learning [ON | OFF] [-Limit integer] [-Verbose {ON | OFF}]
```

Description

Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.

The `set_static_learning` command controls whether the tool performs the learning analysis immediately after design flattening. The tools use the learned behavior for intelligent decision making in later processes, such as ATPG and DRC.

When static learning analysis is on, the additional learned behavior focuses primarily on bus gates. Static learning analysis also allows the test pattern generation process to immediately recognize conflicts and restricted decisions on ATPG constraints that result from the gate assignments. By enabling static learning, you prevent the tools from unnecessarily remaking many decisions, thereby improving the ATPG performance.

For more information about the learning analysis, refer to “[Learning Analysis](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- ON|OFF

An optional literal that disables additional static learning analysis. The default enables the tool to perform additional learning analysis, but you may want to disable to save time if you are not going to be running ATPG.

- -Limit *integer*

An optional switch and integer pair that specifies a single gate simulation activity threshold. When the tool reaches that threshold, it discontinues learning on gates in that design region. You would use the `-Limit` switch for performance reasons. The default value for the *integer* option is 1000.

Caution

 While changing the learning limit increases performance for some designs, it significantly decreases performance for a vast majority of designs. It is very unusual to need to change the learning limit and is therefore not recommended.

- **-Verbose {ON | OFF}**

An optional switch and literal pair that enables additional detailed reporting of analysis results, including the numbers of equivalent gates, classes, implications, and tied gates.

Examples

The following example first enables access to the learned information and then enables the tool to perform additional learning analysis before design flattening:

```
set_learn_report on
set_static_learning on -limit 500
set_system_mode analysis
```

Related Topics

[set_learn_report](#)

set_system_mode

Context: all contexts

Mode: all modes

Specifies the operational state you want the tool to enter.

Usage

```
set_system_mode {setup|insertion | analysis} [-force]
```

Description

Specifies the operational state you want the tool to enter.

The default system mode after tool invocation is setup. The context the tool is in determines which system mode you can specify with the set_system_mode command. See the *Tessent Shell User's Manual* for more information about [contexts and system modes](#).

If you attempt to change back to setup mode and have generated patterns, the tool prompts you to save the patterns with an error message similar to the following:

```
// Error: Current test pattern set has not been saved. Save the  
// patterns or use the -force switch.
```

Arguments

- **setup**

A required literal that specifies to enter Tessent Shell setup mode. This is the default.

- **insertion**

A required literal that specifies to enter Tessent Shell insertion mode.

- **analysis**

A required literal that specifies to enter Tessent Shell analysis mode.

- **-Force**

An optional switch that suppresses the errors that are normally generated when exiting analysis or insertion system modes with unsaved data. Instead of issuing an error, the tool silently deletes unsaved data.

You can also use the -force switch in the following situations:

- When you are in the “patterns -ijtag” context and try to move out of analysis system mode without saving your pattern sets, you cannot change the system mode until you either save the pattern sets or use the -force switch to discard all unsaved pattern sets.
- When you are in insertion system mode and in the rtl context, if you try to change to setup system mode without first saving edited or created modules, you cannot

change the system mode until you either save the modules or use the -force switch to discard the unsaved modules.

Examples

Example 1

The following example changes the system mode so you can perform an ATPG run:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 indata2 outdata4
set_system_mode analysis
create_patterns
```

Example 2

The following example changes the system mode so that you can perform scan stitching:

```
add_tied_signals 1 vcc
add_tied_signals 0 vss
add_clocks 0 clock
set_system_mode analysis
report_scan_elements
insert_test_logic
```

Example 3

The following example changes the system mode so that you can perform an insertion run:

```
set_context dft -no_rtl
read_verilog mydesign.v
read_cell_library tessent_cell.tcellib
set_current_design
set_system_mode insertion
create_instance u1/myInst -of_module myModule
```

Related Topics

[get_system_mode](#)

set_tcl_shell_options

Context: all contexts

Mode: setup

Specifies options for the Tcl shell.

Usage

```
set_tcl_shell_options [-abort_dofile_on_error {off | on | exit}]  
                      [-legacy_dofile_comments {off | on}]  
                      [-legacy_commands_with_spaces {off | on}]  
                      [-minimum_command_typing {off | on}]  
                      [-change_no_result_warnings_to_errors {off | on}]
```

Description

Specifies options for the Tcl shell. You can use this command to control features that provide backward compatibility to old dofiles.

Arguments

- **-abort_dofile_on_error {off | on | exit}**

An optional argument that controls whether the tool aborts or continues dofile execution if it detects an error condition. This switch supersedes the set_dofile_abort command and only affects how the error is handled when it propagates up the nested command stack and reaches the dofile level.

Caution

 Use caution when specifying “-abort_dofile_on_error off” because it traps all potential error conditions. Preferably, you should trap error conditions of specific commands by using the Tcl built-in catch command or the Tesseract catch_output command.

- **-legacy_dofile_comments {off | on}**

An optional argument that controls the treatment of “//” as a comment prefix. If treating “//” as a comment prefix causes issues use “set_tcl_shell_options -legacy_dofile_comments off” to disable it. For new dofiles, you should always use the “#” as a comment prefix.

- **-legacy_commands_with_spaces {off | on}**

An optional argument that controls whether the tool accepts legacy, space-separated commands or not.

- **-minimum_command_typing {off | on}**

An optional argument that controls whether the tool accepts abbreviated commands or not. Using abbreviated commands in your dofile carries the risk that future tool versions will not accept the abbreviations anymore because of ambiguities with new commands.

Specify off for this option to ensure that you always type the full name of a command. The scripts can also run faster when commands are typed in full.

- **-change_no_result_warnings_to_errors {off | on}**

An optional argument that controls whether the tool flags a no result returned from an introspection command as an error or a warning.

The default setting is on, which means that the tool flags as an error a no result returned from an introspection command, unless you specify the -silent switch.

Setting this switch to off instructs the tool to flag with a warning a returned no result list or collection from an introspection command. However, this option introduces a risk if your scripts cannot deal with empty collections correctly. As this may result in incorrect/unintended behavior, it is not recommended.

It is recommended that you ensure your scripts can properly handle an empty collection when it is possible that a get_* introspection command finds nothing. For example, if you use the following, then it must be guaranteed that “[get_instances XXX](#)” command will find an instance:

```
get_modules -of_instance [get_instances XXX]
```

If it is possible for the get_instances command to return an empty collection, use the -silent switch and determine the size of the returned collection before passing it to other procedures or Tesseract Shell commands:

```
set inst [get_instances XXX -silent]
if {[sizeof_collection $inst] > 0} {
    my_procedure $inst
    set mods [get_modules -of_instances $inst]
    ...
}
```

The additional check eliminates any risk of unexpected behavior if the procedure or the command processing the collection cannot correctly handle an empty collection.

Do not use the -silent switch if you expect a command to always return a non-empty collection, and your script does not check the size of the returned collection as shown above.

Using the -silent switch in such cases can cause a valid error message to go unnoticed and result in undesired behavior.

Related Topics

[get_tcl_shell_option](#)

[report_tcl_shell_options](#)

set_test_end_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: all modes

Adds an iCall to the start of the test_end procedure.

Usage

```
set_test_end_icall {[icl_instance_path.]iproc_name [arguments ...]} [[-merge] -append]  
[-non_retargetable]
```

Description

Adds an iCall to the start of the test_end procedure. The set_test_end_icall command adds an iCall to the start of the test_end procedure.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc; preceded by the optional ICL instance path and separated by a period; and one or more optional arguments to the iProc.

If you specify the -append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order the iCall was appended.

If you issue the command without the -append switch, the tool overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

Arguments

- {[icl_instance_path.]**iproc_name** [arguments ...]}

The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- -append

An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.

- -merge

An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the [iMerge](#) command and cannot be unspecified.

Refer to “[iMerge Conflict Reporting](#)” in the *Tessent IJTAG User’s Manual* for more information on the -error_on_conflict option.

The merged iCall commands can be specified in the test_setup/test_end procedures using the tesson shell commands set_test_setup_icall/set_test_end_icall. Later, when you use the [report_procedures](#) command, the merged iCall commands of the test_setup/test_end procedures will appear as follows:

```
iMerge  
  iCall iProcName iProcArgs  
  ....  
iMerge -end;
```

The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

- [-non_retargetable](#)

An optional switch that allows adding iCalls that will not be retargeted at the next level. Use this switch to apply the iCall at the current (core) level and so it is not forwarded when the core is embedded in a design.

Examples

In the following example, the iCall commands specified using the -merge option in the test_end file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_end_icall iProc1 -append  
set_test_end_icall iProc2 -append  
set_test_end_icall iProc3 -merge -append  
set_test_end_icall iProc4 -merge -append
```

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_end =  
  iCall iProc1;  
  iMerge =  
    iCall iProc2;  
    iCall iProc3;  
    iCall iProc4;  
  end;  
end;
```

Related Topics

[get_test_end_icall_list](#)
[get_test_setup_icall_list](#)
[iCall](#)
[set_test_setup_icall](#)

set_test_logic

Context: dft -scan, dft -test_points

Mode: setup

Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.

Usage

```
set_test_logic {-Set {ON | Off} | -REset {ON | Off} | -Clock {ON | Off}  
| -RAm {ON | Off} | -C6 {ON | Off}}...
```

Description

Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.

Before inserting scan chains, you must specify the cell models with the [add_cell_models](#) command.

By default, no test logic is inserted on control signals. You can use the [report_environment](#) command to display the current test logic settings.

If all enable signals of a bus are driven purely by combinational logic, no additional test logic is needed to prevent contention due to scan shifting.

Arguments

- **-Set ON | Off**

A switch and literal pair that enables the insertion of test logic to make set signals controllable. The default setting is off.

- **-REset ON | Off**

A switch and literal pair that enables the insertion of test logic to make reset signals controllable. The default setting is off.

- **-Clock ON | Off**

A switch and literal pair that enables the insertion of test logic to make clock signals controllable. The default setting is off.

- **-RAm ON | Off**

A switch and literal pair that enables the insertion of test logic to make read and write signals controllable. The default setting is off.

This option does not affect the set and reset signals of the RAM. You should use the `edge_trigger` attribute in the library definition of the RAM to specify inserting test logic to control these signals. For more information, see the “[Attributes of RAM/ROM Primitives](#)” section of the *Tessent Cell Library Manual*.

- **-C6 {ON | OFF}**

A switch and literal pair that, when enabled, instructs the tool to add hardware to *fix* identified C6 violations. The tool fixes a C6 violation by inserting a multiplexer into the data path that feeds the flip-flop generating the violation. The test mode input of the multiplexer is driven by a nearby scan flip-flop. When a top-level clock feeds the data input of many flip-flops (which results in multiple C6 violations), the multiplexer is inserted at the highest possible level of hierarchy. The driving scan cells are selected to be in the same clock domain as the flip-flops generating the C6 violations. This setting is off by default.

Note

This option cannot be used when the “`set_drc_handling -Conservative`” switch is enabled.

Examples

The following example checks the set and clock signals of uncontrollable memory elements and makes them controllable with the addition of test logic:

```
add_clocks 0 clk
set_test_logic -set on -clock on
set_system_mode analysis
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Topics

[add_cell_models](#)
[delete_cell_models](#)
[report_cell_models](#)
[set_latch_handling](#)

set_test_point_analysis_options

Context: dft -test_points

Mode: setup, analysis

Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.

Usage

For VersaPoint test points when test point type is set to lbist_test_coverage:

```
set_test_point_analysis_options
  [-total_number integer / integer%] |
  [-control_points_number integer / integer% -observe_points_number integer / integer%] |
  [-test_coverage_target percentage] |
  [-pattern_count_target integer] |
  [-max_control_points_per_path integer] |
  [-exclude_cross_domain_paths on | off] |
  [-observe_primary_outputs on | off] |
  [-exclude_clock_domains off | object_spec] |
  [-allow_in_xbounding_regions on | off]
```

For VersaPoint test points when test point type is set to edt_pattern_count:

```
set_test_point_analysis_options
  [-total_number integer | integer%] |
  [-control_points_number integer / integer% -observe_points_number integer / integer%] |
  [-max_control_points_per_path integer] |
  [-exclude_cross_domain_paths on | off] |
  [-exclude_clock_domains off | object_spec] |
  [-allow_in_xbounding_regions on | off]
```

For VersaPoint test points when test point type is set to {edt_pattern_count lbist_test_coverage}:

```
set_test_point_analysis_options
  [-total_number integer / integer%] |
  [-control_points_number integer / integer% -observe_points_number integer / integer%] |
  [-test_coverage_target percentage] |
  [-pattern_count_target integer] |
  [-max_control_points_per_path integer] |
  [-exclude_cross_domain_paths on | off] |
  [-observe_primary_outputs on | off] |
  [-exclude_clock_domains off | object_spec] |
  [-allow_in_xbounding_regions on | off]
```

Description

Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.

Not all gates in the design will be considered by the tool for inserting test points. For various reasons, test points will not be inserted on gates in the clock tree or in the scan path, gates in protected blocks or gates where the user has explicitly requested that no test points should be inserted. If there are many locations where test points cannot be inserted the tool may not be able to find the best locations for test points. This may increase the number of test points that are needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20% of the gate-pins the tool will generate a warning. The warning specifies the number of these locations as well as the main reason(s) why test points cannot be inserted at these locations.

By default, if you do not specify the maximum number of test points using the `-total_number`, `-control_points_number` or `-observe_points_number` switches, the tool will assume the total number of test points to be 1% of the identified memory elements in the design for EDT test points. When the test point type `lbist_test_coverage` is used, the tool will assume the number of `test_points` to be 10000000.

Please refer to the “[Test Points for Improving the Test Coverage of Deterministic Patterns](#)” in the *Tessent Scan and ATPG User’s Manual* for generating test points that improve the testability of a subset of faults.

Arguments

- `-total_number integer / integer%`

An optional switch and integer or integer percentage that specifies the maximum number of test points to be inserted as an integer or integer percentage of the total number of flops in the design, with no maximum as default. This maximum only applies to the test points automatically generated by the tool using the command `analyze_test_points`.

Note



The integer percentage must be specified as an integer followed by %.

- `-control_points_number integer / integer%`

An optional switch and integer or integer percentage that specifies the maximum number of control points to be inserted as an integer or integer percentage of the total number of flops in the design, with no maximum as default.

This maximum only applies to the control points automatically generated by the tool using the `analyze_test_points` command. Issuing these two commands is sub-optimal; the preferred flow is to let the tool automatically decide the test point distribution.

- **-observe_points_number integer / integer%**

An optional switch and integer or integer percentage that specifies the maximum number of observe points to be inserted as an integer or integer percentage of the total number of flops in the design, with no maximum as default.

This maximum only applies to the observe points automatically generated by the tool using the analyze_test_points command. Issuing these two commands is suboptimal; the preferred flow is to let the tool automatically decide the test point distribution.

Note



The number of control points and the number of observe points must both be specified in the same manner, that is, as an integer or integer percentage.

- **-test_coverage_target percentage**

An optional switch and integer that specifies the target fault coverage to be achieved by the number of random patterns. The tool will insert test points until either the maximum number of test points is reached or the desired fault coverage with the target number of random patterns is achieved. The default is 99%.

When you perform test point analysis for a set of target faults (instead of all faults), the default value is 100%.

Note



This option is not relevant when test point type is set to edt_pattern_count.

- **-pattern_count_target integer**

An optional switch and integer that specifies the number of random patterns to be applied to the design. Fault coverage estimations will be done using this parameter as reference. Default is 100,000.

When you perform test point analysis for a set of target faults (instead of all faults), the default value is 1000.

Note



This option is not relevant when test point type is set to edt_pattern_count.

- **-max_control_points_per_path integer**

An optional switch and integer that specifies the maximum number of control points to be inserted on a single path. The default maximum number of control points per path is five.

- **-exclude_cross_domain_paths on | off**

An optional switch and literal that prevents the tool from inserting test points in inter-clock domains. The default is off.

- **-observe_primary_outputs on | off**

An optional switch and literal pair that allows you to control the insertion of observe points at primary outputs. The default is off - no observe points are inserted at primary outputs.

Note

 This option is not relevant when test point type is set to edt_pattern_count.

If you set this switch to on, an observe point is inserted at each primary output where at least three previously unobserved cells can be observed.

If you have specified a target set of faults, the tool will by default insert an observe point at each primary output where at least ten user-specified faults are observed by the inserted observe point that would otherwise not be observable.

When you are in Analysis mode and have selected test points with the analyze_test_points command, if you try to change the setting of the -observe_primary_outputs switch from on to off, you will get an error message that you need to first delete all test points before the -observe_primary_outputs switch can be changed. If you try to change the setting of the -observe_primary_outputs switch from off to on, however, and test points have already been selected, you will get a warning that some of the test points may no longer be optimal after adding test points to primary outputs, and for best results you should delete all test points before proceeding with the analyze_test_points command.

Note

 The number of observe points inserted on primary outputs is independent of the total number of test points or the number of observe points that you may specify with the -total_number or -observe_point options of the set_test_point_analysis_options command. The total number of test points that are reported by report_test_points will include test points inserted at the primary outputs because the -observe_primary_outputs option was used, as well as those identified using our standard algorithm. Therefore, the total number may exceed the user specified threshold.

- **-exclude_clock_domains off | object_spec**

An optional switch and literal that prevents the tool from inserting test points at any gate within a specified clock domain. The default is off.

The value of object_spec can be a Tcl list of one or more object names or a collection of one or more objects. Typically, these objects are primary input pins that have been specified as clocks by the add_clocks command. The default test_clock or the test clock specified with the set_scan_signals command is also an acceptable clock object.

When you use the “set_test_point_analysis_options -exclude_clock_domains” command multiple times, the latest specification of the “-exclude_clock_domains” switch overrides all previous specifications. Using the “exclude_clock_domains” switch with the “off” argument (“-exclude_clock_domains off”) deletes all excluded clock domains.

By using this switch, you can prevent test points from the following locations:

- Any gate that receives data that is launched from a flip-flop that is clocked by an excluded clock.
- Any gate whose output value is captured at the data port of a flip-flop that is clocked by an excluded clock.

If you want to allow test points at some gates in the excluded clock domains, set the “no_control_point”/“no_observe_point” attributes to false or use the delete_notest_point command to overwrite the “-exclude_clock_domain” switch. The set_attribute_value and delete_notest_point commands have a higher priority than the “-exclude_clock_domains” switch. These specifications (delete_notest_points command, “no_control_point”/“no_observe_point” attribute value, “-exclude_clock_domains” switch) can be entered in any order and can be entered in setup mode or analysis mode. The locations where no test points are allowed will be consistent irrespective of the order of the commands and the time (setup or analysis mode) when they are entered.

In both setup and analysis modes, the set_test_point_analysis_options command with no options will report the excluded clock domains along with the settings of the other switches. In analysis mode the “no_control_point”/“no_observe_point” attributes of the gates in the excluded clock domains are set to true. Therefore, the actual gates that are in these excluded clock domains can be reported or introspected in analysis mode. For example, in analysis mode the “report_notest_points” command can be used to show where test points are not allowed, including the locations in the excluded clock domains.

In analysis mode, if you have selected test points with the analyze_test_points command and then want to change the excluded clock domains, an error will be reported and you will be advised that all test points must be deleted before the excluded clock domains can be changed.

When you specify an excluded clock domain, the tool will first check whether the specified pin is indeed a clock. That is, it will verify that the excluded clock domain has previously been specified as a clock with the add_clocks command (the default test_clk or the test clock that is specified with the set_scan_signals command will also be accepted). An error will be reported if the pin is not a clock. In analysis mode (or during the transition to analysis mode) the tool will also verify whether the excluded clock is actually clocking any flip-flops. A warning will be reported if an excluded clock pin connects to the set/reset inputs of one or more flops. The warning will mention the clock pins that connect to a set/reset pin, but the flops with a set/reset pin that is connected to an excluded clock will not be listed. The connections to set/reset inputs will be ignored. Only flip-flops that are actually clocked by an excluded clock pin will be used to identify gates in the excluded clock domains.

Note

 When you exclude a top clock domain, the clock domains derived from it are not automatically excluded. This gives you the flexibility to exclude the top clock domain without excluding all derived clock domains. However, the tool will issue a warning when a top clock domain is excluded but one or more of the derived clock domains are not excluded.

- -allow_in_xbounding_regions on | off

An optional switch and literal that specifies whether or not to allow the insertion of test points in the X-bounding regions identified during X-bounding analysis. Set this switch to **on** to allow the tool to insert test points in the X-bounding regions. The default value is **off**.

when the test point type is set to only lbist_test_coverage, and **on** any time the test point type edt_pattern_count is used.

Note

 During X-bounding, the tool adds muxes or other logic to block X-sources at specific nets within the design. This blocking logic is placed as close to the X-source as possible, but may not be directly at the source. X-bounding regions are defined as any nets or pins that are in the path of the X-source prior to reaching the X-bounding hardware. These nets and pins may attain an X state even after the X-bounding hardware has been added.

Examples

Example 1

The following example specifies that enough test points should be generated to achieve 99.5 percent fault coverage, but not more than a maximum number of 3000 test points, for a design that will be tested with a set of 131072 random patterns:

```
set_test_point_analysis_options -test_coverage_target 99.5 -total 3000 \
-patterncount_target 131072
```

The following is a snippet from the log file that shows the test coverage reporting before and after selecting test points:

```

// command: analyze_test_points
// Identifying locations of x-bounding muxes prior to test point analysis
//
// Test Coverage Report before Test Point Analysis
// -----
// Target number of random patterns          131072
//
// Total Number of Faults                  15577
//   Testable Faults                      15063  ( 96.70%)
//     Logic Bist Testable                14970  ( 96.10%)
//     Blocked by xbounding               0      ( 0.00%)
//     Uncontrollable/Unobservable       93     ( 0.60%)
//
// Estimated Maximum Test Coverage        99.38%
// Estimated Test Coverage (pre test points) 96.19%
// Estimated Relevant Test Coverage (pre test points) 96.24%
//
// Warning: Test points cannot be inserted at 4033 (33.6%) gate-pins.
// This may increase the number of test points that is needed to reach the
// fault coverage target.
// 3929 (32.7%) gate-pins are located on clock lines or on the scan path.
//
// Incremental Relevant Test Coverage Report
// -----
// TPs 50 =    21 (CP) +    29 (OP), TC 99.41
// TPs 100 =   42 (CP) +   58 (OP), TC 99.41
// TPs 150 =   62 (CP) +   88 (OP), TC 99.42
// (snipped)
// TPs 1150 = 128 (CP) + 1022 (OP), TC 99.49
// TPs 1200 = 135 (CP) + 1065 (OP), TC 99.49
// TPs 1250 = 148 (CP) + 1102 (OP), TC 99.50
//
// Test point analysis completed: target estimated test coverage has been
// achieved.
// Inserted Test Points                 1297
// Control Points                      157
// Observe Points                      1140
//
// Test Coverage Report after Test Point Analysis
// -----
// Target number of random patterns          131072
//
// Total Number of Faults                  18042
//   Testable Faults                      17528  ( 97.15%)
//     Logic Bist Testable                17435  ( 96.64%)
//     Blocked by xbounding               0      ( 0.00%)
//     Uncontrollable/Unobservable       93     ( 0.52%)
//
// Estimated Test Coverage (post test points) 99.03%
// Estimated Relevant Test Coverage (post test points) 99.50%
```

Note

- The “Total Number of Faults”, the number of “Testable Faults” and the number of “Logic Bist Testable” faults are larger after test point insertion as these numbers include the (testable) faults at the new test point logic.
-

Example 2

The following example specifies that enough test points should be generated to achieve 99 percent fault coverage (default), but not more than a maximum number of 3000 control points, and a maximum of 2000 observe points:

```
set_system_mode analysis
set_test_point_analysis_options -control_points 3000 -observe_points 2000
analyze_test_points
report_test_points
insert_test_logic
```

Example 3

The following example specifies excluded clock domains and uses the **set_test_point_analysis_options** command to display the excluded clock domains. Then it adds a few more clock domains and displays the excluded clock domains again. The example uses a collection named \$some_clocks which is comprised of two clock pins and a collection named \$more_clocks which is comprised of four clock pins:

```
set_test_point_analysis_options -exclude_clock_domains $some_clocks
set_test_point_analysis_options

// Maximum Number of Test Points      :      50
// Maximum Number of Control Points   :      50
// Maximum Number of Observe Points  :      50
// Target Test Coverage               :    99.00
// Target Pattern Count              :    1000
// Maximum Control Points per Path   : 65535
// Exclude Cross Domain Paths       :    off
// Excluded_Clock_Domains           : fast_clock1
//                                         fast_clock2
//                                        

set_test_point_analysis_options -exclude_clock_domains $more_clocks
set_test_point_analysis_options

// Maximum Number of Test Points      :      50
// Maximum Number of Control Points   :      50
// Maximum Number of Observe Points  :      50
// Target Test Coverage               :    99.00
// Target Pattern Count              :    1000
// Maximum Control Points per Path   : 65535
// Exclude Cross Domain Paths       :    off
// Excluded_Clock_Domains           : fast_clock1
//                                         fast_clock2
//                                         fast_clock3
//                                         fast_clock4
```

Example 4

This example specifies that no control points can be inserted in clock domain fast_clock1. The report_notest_points command shows the locations in the excluded clock domain (gates U10 to U90). Next the no_control_point attributes for the pins on gate U10 are reset to false. After that the report_notest_points command shows all gates in the excluded clock domain. Note that gate U10 is no longer excluded:

```
set_test_point_analysis_options -exclude_clock_domains fast_clock1
set_system_mode analysis
report_notest_points

/U10/A
/U10/B
/U10/Z
/U11/A
/U11/B
/U11/Z
...
/U90/Z

set_attribute_value U10 -name no_control_point -value false
{U10}

report_notest_points

/U11/A
/U11/B
/U11/Z
...
/U90/Z
```

Example 5

In this example, you specify an excluded clock domain after test points have been inserted. An error is reported as this is not allowed until all test points have been deleted:

```
set_system_mode analysis
analyze_test_points

// ...
// Test point analysis completed: target estimated test coverage has been
achieved.
// Inserted Test Points 2
// Control Points 2
// Observe Points 0
// ...

set_test_point_analysis_options -exclude_clock_domains fast_clock1

// Error: Excluded clock domains cannot be specified when test points
have already been inserted.
// Please delete all test points before proceeding with specifying
(an) excluded clock domain(s).
```

Example 6

The number of control points and observe points must both be specified as an integer or as an integer percentage. The following command will result in an error.

```
set_test_point_analysis_options -control_points 30 -observe_points 5%
```

Related Topics

[analyze_test_points](#)

[report_test_points](#)

[set_test_point_insertion_options](#)

[set_test_point_types](#)

[write_scan_setup](#)

set_test_point_insertion_options

Context: dft -test_points

Mode: setup, analysis

Prerequisites: None

Sets parameters related to test point insertion.

Usage

```
set_test_point_insertion_options
  [-control_point_enable pin_path_name]
  [-observe_point_enable pin_path_name]
  [-observe_point_share integer]
  [-test_point_clock system_clocks | always_tclk | tclk_as_needed ]
  [-prefer_scan_cells on | off]
```

Description

Sets parameters related to test point insertion.

These parameters will impact the structure of the test logic that is generated when the [insert_test_logic](#) command is issued.

Arguments

- **-control_point_enable *pin_path_name***

An optional switch and string pair that specifies the pin and path name of the enable signal for control points. In its absence, the name lbist_en will be used at the top level.

- **-observe_point_enable *pin_path_name***

An optional switch and string pair that specifies the pin and path name of the enable for observe points. In its absence, the name lbist_en will be used at the top level.

- **-observe_point_share *integer***

An optional switch and integer pair that specifies how many observe points will be observed by one scan cell. The default is one observe point to one destination scan cell.

Note



Mentor Graphics recommends not to share multiple observe points per scan cell when test point type is set to atpg, even though it is supported by the tool. The reason is that it impacts the pattern count reduction during ATPG.

-
- **-test_point_clock *system_clocks* | *always_tclk* | *tclk_as_needed***

An optional switch and literal pair that specifies what clock to use to drive scan cells for test points.

The valid literals are as follows:

system_clocks — A literal that specifies that all test points should be controlled by system clocks. Normally, the test point clock is selected based on analysis of the fanin/fanout cone. Instead of using `test_clock`, when this analysis fails to identify a suitable clock, the tool will select the system clock that drives the most scannable cells within the design. This is the default.

Note

 When selecting an existing system clock, the tool will give preference to clocks that do not drive the set or reset port of any scannable flops. The test clock must pulse during shift, so we do not want to choose a clock that is also connected to the set/reset port of scannable flops since this would cause S2 violations for those flops.

always_tclk — A literal that specifies to use the user_defined `test_clock` signal for all sequential elements that are inserted during test point insertion. When this option is specified, the tool takes the global clock name from the `-tclk` option of the `set_scan_signals` command.

tclk_as_needed — A literal that specifies to use the user_defined `test_clock` when no other option is found in the fanin or fanout cone, or for observe points, when both the fanin and fanout cone connect to multiple clock domains and false or multicycle paths have been defined.

- **-prefer_scan_cells on | off**

An optional switch and literal that controls the search order that is used to identify a scan cell that will be inserted to control or observe a test point. The default is on. The tool will first look for a scan cell, and will only use a non-scan cell if a scan cell is not found. The scan cell can be automatically identified by the tool and reported by “`report_cell_models`” or manually specified by the “`add_cell_models`” command.

When `-prefer_scan_cells` is off, the tool will first look for a non_scan cell, and will use a scan cell only when the “`report_cell_models`” command reports that all available cell models are scan cells.

Note

 Prior to the v2014.4 release, the tool would insert a non-scan cell by default.

Examples

Example 1

The following example uses the “`-observe_point_share`” switch to determine the size of the XOR tree that is generated when the tool merges several observe points together before capturing the value into either a new or existing scan cell. This example XOR’s together no more than 7 observe points before feeding the output of the XOR tree into a scan cell.

```
set_test_point_insertion_options -observe_point_share 7
```

Example 2

The following example specifies the names of enables for control points and observe points respectively. It also specifies that the global clock that was specified with the `set_scan_signals` command ('`test_clk`') is to be used to drive the scan cells for the test points:

```
set_scan_signals -tclk test_clk
...
set_test_point_insertion_options -control_point_enable control_enable
set_test_point_insertion_options -observe_point_enable observe_enable
set_test_point_insertion_options -test_point_clock always_tclk
```

Example 3

The following example generates a maximum of 5000 test points. Then it specifies the names of enables for control points and observe points respectively, and it inserts the test points into the netlist:

```
set_system_mode analysis
set_test_point_analysis -total_number 5000
analyze_test_points
report_test_points
set_test_point_insertion_options -control_point_enable cp_enable \
-observe_point_enable op_enable
insert_test_logic
```

Related Topics

[add_cell_models](#)
[analyze_test_points](#)
[insert_test_logic](#)
[set_test_point_analysis_options](#)
[set_test_point_types](#)
[write_scan_setup](#)

[set_test_point_types](#)

Context: dft -test_points

Mode: setup

Specifies the type of test points to insert in the design.

Usage

```
set_test_point_types [lbist_test_coverage | edt_pattern_count |  
 {lbist_test_coverage edt_pattern_count}]
```

Description

Specifies the type of test points to insert in the design. If no argument is specified, the default is edt_pattern_count test points.

Arguments

- lbist_test_coverage

An optional switch that specifies that the test points inserted will be for improving random pattern test coverage.

When you specify this switch:

- The tool will perform X-bounding analysis by default. Any node with an Xsource will be assumed to be bounded and therefore will be fully controllable.
- All the faults behind X-bounding logic will be ignored during analysis. Both incremental test coverage and test coverage estimates will be written out before and after test point analysis.
- The targeted pattern counts and test coverage thresholds will be taken into account.
- The default pattern count is 10000 and test coverage is 99.

If you provide a fault list, test point analysis will try to improve ATPG test coverage for the fault list provided. In this case, X-bounding analysis will not be done by default. The default pattern count will be 1000 and default coverage will be 100.

- edt_pattern_count

An optional switch that specifies that the test points inserted will be for reducing deterministic pattern count.

When you specify this switch:

- The tool will not perform X-bounding analysis by default.
- No incremental test coverage and test coverage estimates are written out.
- If a fault list is provided in conjunction with this option, this will be the fault list used to insert the specified number of test points for ATPG pattern count reduction, instead of the full fault list, which may result in non-ideal results.

- **lbist_test_coverage edt_pattern_count**

If you specify both types of test points, the tool identifies test points to reduce both pattern count reduction and improve random pattern testability.

- The tool will not perform X-bounding analysis by default.
- Both incremental test coverage and test coverage estimates will be written out before and after test point analysis.
- The targeted pattern counts and test coverage thresholds will be taken into account.
- If a fault list is provided in conjunction with both these options specified together, this will be the fault list used to insert the specified number of test points for ATPG pattern count reduction and LBIST coverage improvement, instead of the full fault list which may result in non-ideal results. Coverage improvement may not be as good as it could be when using just the **lbist_test_coverage** option alone.
- The default pattern count is 10000 and test coverage is 99.

Examples

The following example sets the test points type for test coverage improvement, performs test point analysis and then reports the test points for your design:

```
set_context dft -test_points -no_rtl
set_test_point_types lbist_test_coverage
set_system_mode analysis
set_test_point_analysis_options -total_number 1000
analyze_test_points
report_test_points
```

Related Topics

[add_control_points](#)
[add_observe_points](#)
[analyze_test_points](#)
[get_test_point_type](#)
[set_test_point_analysis_options](#)
[set_test_point_insertion_options](#)
[write_scan_setup](#)

[set_test_setup_icall](#)

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: all modes

Adds an iCall to the end of the test_setup procedure.

Usage

```
set_test_setup_icall {[icl_instance_path.]iproc_name [arguments ...]}  
  [[-merge] -append] {[[-front] | [-end] [-non_retargetable]}}
```

Description

The set_test_setup_icall command adds an iCall to the end of the test_setup procedure.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

If you specify the –append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order they were appended.

If you issue the command without the -append switch, it overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

If the test_setup procedure contains iCalls, then Tesson Shell infers input constraints on the ports with ICL port function DataInPort. For more details, see the description of the [add_input_constraints](#) command.

To keep the iCalls of the set_test_setup_icall command in the correct order, the iCall can be added to the front, middle, or end list.

The iCalls from the front list will be executed first. The iCalls from the middle list will be executed second. The iCalls from the end list will be executed last.

By default, the targeted list that the new iCall is added to is the middle list, but you can override this default with the following options:

- -front — This option adds the iCall to the front list.
- -end — This option adds the iCall to the end list.

The -append switch adds the iCall to the end of the targeted list. If you add an iCall without using the -append switch, then all the previous iCalls of the targeted list will be cleared. Note: This only clears all the iCalls of the targeted list, not all the iCalls of all three lists. When you

add an iCall with the -merge switch, then this iCall will be merge with the previous iCall on the same targeted list if a previous iCall exists.

Additionally, you can add [report_test_setup_icall](#) and [report_test_end_icall](#) commands to report iCalls configuration in test_setup and test_end procedures.

If you need to clear the test_setup_icall list, specify either quotes ("") or braces ({{}}) as an argument to the set_test_setup_icall command. Both arguments work the same and will clear the list.

Arguments

- {[*icl_instance_path.*]**iProc_name** [*arguments ...*]}

The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- **-append**

An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.
- **-merge**

An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the [iMerge](#) command and cannot be unspecified.

Refer to “[iMerge Conflict Reporting](#)” in the *Tessent IJTAG User’s Manual* for more information on the -error_on_conflict option.

The merged iCall commands can be specified in the test_setup/test_end procedures using the tessent shell commands [set_test_setup_icall](#)/[set_test_end_icall](#). Later, when you use the tessent shell command [report_procedures](#) (available in analysis mode), the merged iCall commands of the test_setup/test_end procedures will appear as follows:

```
iMerge
    iCall iProcName iProcArgs
    ...
iMerge -end;
```

The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

- **-front**

An optional switch that instructs the tool to add the specified iCall to the first list which will cause it to run first.

- **-end**
An optional switch that instructs the tool to add the specified iCall to the first list which will cause it to execute last.
- **-non_retargetable**
An optional switch that allows adding iCalls that will not be retargeted at the next level. Use this switch to apply the iCall at the current (core) level and so it is not forwarded when the core is embedded in a design. This is especially needed to handle BISR programming when memories are used during scan test.

Examples

Example 1

The following example adds an iCall to the end of the test_setup procedure, then appends and merges a second iCall. The get_test_setup_icall_list command displays the added iCalls.

```
set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on
                     edt_bypass off"

set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on
                     edt_bypass off" -append -merge

get_test_setup_icall_list

{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Example 2

In the following example, the iCall commands specified using the -merge option in the test_setup file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_setup_icall iProc1 -append
set_test_setup_icall iProc2 -append
set_test_setup_icall iProc3 -merge -append
set_test_setup_icall iProc4 -merge -append
```

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_setup =
  iCall iProc1;
  iMerge =
    iCall iProc2;
    iCall iProc3;
    iCall iProc4;
  end;
end;
```

Example 3

The following command sequence constructs three lists of iCalls. The report_test_setup_icall command is then used to report all of the existing iCalls that were specified with set_test_setup_icall:

```
set_test_setup_icall userProc_1 -front
set_test_setup_icall userProc_2 -front -append
set_test_setup_icall userProc_3 -append
set_test_setup_icall userProc_4 -append -merge
set_test_setup_icall userProc_5 -append -end
set_test_setup_icall userProc_6 -append -end
set_test_setup_icall toolProc_1 -front -append
set_test_setup_icall toolProc_2 -append
set_test_setup_icall toolProc_3 -append -merge

// first list of iCalls with -front:
// userProc_1
// userProc_2
// toolProc_1

// second list of iCalls without -front and -end
// userProc_3
// userProc_4 -merge
// toolProc_2
// toolProc_3 -merge

// third list of iCalls with -end
// userProc_5
// userProc_6
report_test_setup_icall

iCall userProc_1
iCall userProc_2
iCall toolProc_1
iMerge -begin
    iCall userProc_3
    iCall userProc_4
iMerge -end
iMerge -begin
    iCall toolProc_2
    iCall toolProc_3
iMerge -end
iCall userProc_5
iCall userProc_6
```

Related Topics

[get_test_end_icall_list](#)
[get_test_setup_icall_list](#)
[iCall](#)
[report_test_end_icall](#)
[report_test_setup_icall](#)

set_test_end_icall

set_testbench_simulation_options

Context: unspecified, all contexts

Mode: all modes

Specifies the default values for a set of options that exist in the [run_testbench_simulations](#) command.

Usage

```
set_testbench_simulation_options [-default_simulator {questa | vcs | incisive}]  
[-simulation_output_directory directory]  
[-parallel_simulations {1 | 2...MAX_INT | maxcpu}]  
[-keep_simulation_data {on | off | on_failure}]  
[-store_simulation_waveforms {on | off}]  
[-simulation_timeout {time | unlimited}]  
[-compilation_options option_list]  
[{questa | vcs | incisive} [-simulator_options option_list] ]  
[-simulation_run_commands simulator_commands]  
[-waveform_configuration_commands simulator_commands] ]
```

Description

Specifies the default values for a set of options that exist in the [run_testbench_simulations](#) command.

This command also configures the default value of the `-simulation_output_directory` list of the [check_testbench_simulations](#) command.

Arguments

- `-default_simulator questa | vcs | incisive`
An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name.
- `-simulation_output_directory simulation_output_directory`
An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name. It also sets the default value of the [check_testbench_simulations](#) `-simulation_output_directory` list option.
- `-parallel_simulations {1 | 2...MAX_INT | maxcpu}`
An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name. *MAX_INT* is the maximum integer value that the host machine allows. The *maxcpu* literal specifies using all available CPUs on the host machine.
- `-keep_simulation_data on | off | on_failure`
An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name.

- **-store_simulation_waveforms on | off**

An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name.

- **-simulation_timeout time | unlimited**

An optional switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name.

- **-compilation_options *option_list***

An optional switch and value pair that specifies a list of compilation options (for example, +initreg) that modify the compilation script during runtime. The *option_list* uses the following syntax:

```
-compilation_options {[global_options] [verilog verilog_options] [vhdl {vhdl_options}]}
```

- The verilog options are passed only to the command that compiles Verilog source files.
- The vhdl options are be passed only to the command that compiles the VHDL source files.
- The *global_options* can also be expressed as “global {*global_options*}”.

The braces are mandatory in order to identify the options preceded by the language or “global”.

See [Example 4](#).

- **questa | vcs | incisive**

A simulator literal that allows you to specify default values for options that pertain to the specified simulator.

- **-simulator_options {-timescale *value* ...}**

An optional simulator-specific switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name. This can only be specified with one simulator literal.

The -timescale *value* option overrides the default timescale for a specific simulator for the following default values:

- incisive — Default timescale is 1ns/1ps

- **-simulation_run_commands *simulator_commands***

An optional simulator-specific switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name. This can only be specified with one simulator literal.

- **-waveform_configuration_commands simulator_command_list**

An optional simulator-specific switch and value pair that, when specified, sets the default value for the [run_testbench_simulations](#) option of the same name. This can only be specified with one simulator literal.

Examples

Example 1

The following command sets the default value of the run_testbench_simulation -keep_simulation_data option to on. It demonstrates using a simulator-independent option.

```
set_testbench_simulation_options -keep_simulation_data on
```

Example 2

The following command sets the default value of the run_testbench_simulation -simulator_options passing the simulator-specific option of -novopt. This is a simulator-specific option so the simulator literal (questa) is specified. The options take effect when you use run_testbench_simulations -simulator questa, or when the default simulator is questa and you specify run_testbench_simulations without the “-simulator” argument.

```
set_testbench_simulation_options questa -simulator_options {-novopt}
```

Example 3

In the following example, the options take effect when you use run_testbench_simulations -simulator incisive, or when the default simulator is incisive and you specify run_testbench_simulations without the “-simulator” argument.

```
set_testbench_simulation_options incisive -simulator_options \  
 {+nctimescale+1ns/100ps +access+rwc+nowarn+CUVWSP +nowarn+CUVWSI  
 +nowarn+CUVWSP}
```

Example 4

The following example demonstrates using the -compilation_options switch. In this example, the [run_testbench_simulations](#) command uses the vcs simulator and the -compilation_options value is set by the [set_testbench_simulation_options](#) command.

```
set_testbench_simulation_options vcs \  
 -compilation_options { global { -kdb } vhdl { -skip_translate_body }  
 verilog { -timescale=1ns/1ps } } -default_simulator vcs  
  
run_testbench_simulations
```

set_tied_signals

Context: all contexts

Mode: setup

Changes the default value for floating pins and floating nets which do not have assigned values.

Usage

`set_tied_signals X | 1 | 0 | Z`

Description

Changes the default value for floating pins and floating nets which do not have assigned values.

The `set_tied_signals` command specifies the default value that the tool ties to all floating nets and floating pins that you do not specify with the `add_tied_signals` command. Upon invocation of the tool, if you do not assign a specific value, the tool assumes the default value is unknown (X).

If the model is already flattened and then you use this command, you must delete and recreate the flat model.

Arguments

- **X**
A literal that ties the floating nets or pins to unknown. This is the default upon invocation of the tool.
- **0**
A literal that ties the floating nets or pins to logic 0 (low to ground).
- **1**
A literal that ties the floating nets or pins to logic 1 (high to voltage source).
- **Z**
A literal that ties the floating nets or pins to high-impedance.

Examples

The following example ties floating net vcc to logic 1, ties the remaining unspecified floating nets and pins to logic 0, then performs an ATPG run:

```
set_tied_signals 0
add_tied_signals 1 vcc
set_system_mode analysis
create_patterns
```

Related Topics

[add_tied_signals](#)

[delete_tied_signals](#)

[report_tied_signals](#)

set_timing_exceptions_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Changes the default handling of timing exception paths (false paths and multicycle paths) read by the [read_sdc](#) command.

Usage

```
set_timing_exceptions_handling [ON | OFF] [-CLOCK_groups {ON | OFF}]
[-ADD_Hold_time_when_no_settings_in_sdc {ON | OFF}] [-LIne_no {ON | OFF}]
[-SETUp {ON | OFF}]
[-MAX_Skew {HALf_cycle | ONE_cycle | MULtiple_cycles}]
[-APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFF}]
[-DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFF}]
[-X_Statistics {ON | OFF}]
[-SLOW_Cycle_mask {ON | OFF}]
[-ALLOW_Invalid_pin_names {ON | OFF}]
[-MERGE_compatible_paths {ON | OFF}]
```

Description

Changes the default handling of timing exception paths (false paths and multicycle paths) read by the [read_sdc](#) command.

When you issue the `set_timing_exceptions_handling` command without options, the command reports the current settings.

Arguments

- **ON | OFF**
A literal that enables or disables handling of timing exceptions on false paths and multicycle paths. This capability is enabled by default upon invocation of the tool. Turning off this switch is useful for debugging low coverage or simulation mismatch because you can quickly verify if the masking values or test coverage loss is due to timing exceptions. Turning off this switch can result in pattern mismatches, so you should not turn off this switch other than for debugging purposes.
- **-CLOCK_groups {ON | OFF}**
A switch and literal pair that allows you to change the clock groups handling in SDC. By default, the tool expands timing exceptions for the clocks in asynchronous clock groups as both setup and hold timing exceptions only when the clocks can be pulsed by ATPG concurrently. If you are not generating any patterns that could pulse asynchronous clocks concurrently during ATPG for the ones defined in the SDC files, you can disable this path expansion by specifying OFF.

If -clock_groups -on is used and there are already false paths derived from existing clock groups, the following message will be reported:

```
// Warning: The clock groups have been expanded into false paths.  
You must delete SDC to remove false paths derived from clock groups.
```

To avoid this issue, use the “set_timing_exceptions_handling -clock_groups -off” command prior to running read_sdc.

- **-ADD_Hold_time_when_no_settings_in_sdc {ON | OFF}**

A switch and literal pair that allows you to treat the timing exceptions to include both setup and hold-time violations when the SDC files specifies neither -setup nor -hold. By default, the tool matches the SDC file used by your static timing analyzer, which is useful if your design must include hold timing exceptions during stuck-at ATPG. When you turn off this switch, the tool assumes that all false paths without -setup or -hold options have only setup timing exceptions.

[Table 6-20](#) further explains how the tool treats timing exception paths based on the setting in the SDC file and the -ADD_Hold_time_when_no_settings_in_sdc switch.

Table 6-20. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch

SDC file switches	Default tool behavior	Tool behavior with switch OFF
none	setup & hold	setup *
-setup	setup	setup
-hold	hold	hold
-setup -hold	setup & hold	setup & hold

* True unless the timing exception path is added across clock domains, in which case the path is treated as both setup & hold. As an example, the following SDC commands are equivalent:

```
set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}}
set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}} -setup -hold
```

- **-LIne_no {ON | OFF}**

An optional switch and literal pair that allows you to turn on reporting of SDC filename and line numbers for timing violations when using the [report_false_paths](#) or the [report_multicycle_paths](#) command. By default, the tool does not report the SDC filename and line numbers.

- **-SETUp {ON | OFF}**

An optional switch and literal pair that disables setup time simulation without having to modify the SDC file. By default, setup time simulation is enabled.

Note that the tool automatically reclassifies AU faults when you change this option, so there is no need to issue a “reset_au_faults -reclassify” command.

- **-MAX_Skew**

An optional switch that specifies the maximum clock skew allowed when simulating the hold time exception. This option impacts the masking effect of most hold time exceptions defined using either the add_false_path command or the hold time exceptions from read_sdc. This option does not impact false paths added with the “add_false_paths -cross_clock_domains” command and the false paths derived from the asynchronous clock groups in SDC files.

HALf_cycle

Specifies half cycle as the maximum clock skew. This is the default value for this option, unless -cross_clock_domains is specified, in which case, the max skew is multiple cycle.

ONE_cycle

Specifies one cycle as the maximum clock skew.

MULtiple_cycles

Specifies multiple cycles as the maximum clock skew.

- **-APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFF}**

An optional switch and literal pair that specifies the behavior of multicycle path (MCP) cycle scaling.

The following text explains the concept of cycle scaling. For synchronous clock domains with different frequencies (either within each source clock or across all source clocks), the tool scales the clocks according to their frequency information provided in the SDC file. In cases when the driving clock of a multicycle path has a different frequency source clock, this switch defines how the tool should determine their relationship. For example, consider a multicycle path with 2 cycles defined with -end in the SDC file. If the frequency of the endpoint driving clock is twice as slow as its source clock, the tool scales the clocking so that the path is treated as a multicycle path with 4 cycles.

This switch has the following options:

WIthin_each_source_clock — Applies the cycle scaling only for the generated clocks defined in SDC using the create_generated_clock command with the same source clock. Treats different source clocks independently, and treats all generated clocks with the same source clock as synchronous clocks. This is the default.

ACross_all_source_clocks — Applies the cycle scaling across all source clocks. This option is applicable when treating all source clocks as synchronous.

OFF — Does not apply the cycle scaling, so the tool uses the cycle count as defined in SDC set_multicycle_path commands.

- **-DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFF}**

An optional switch and literal pair that allows the tool to ignore SDC-generated clocks from which to derive the fault detection cycles. By default, the tool allows ATPG and the fault simulator to consider the transition faults in slower clock domains to be detected with multiple activation cycles without needing to define multicycle paths. For example, if a

transition fault is on a divide-by-2 clock domain, the fault can be activated for up to 2 cycles before capture. The switch default is ON, which instructs ATPG and the fault simulator to consider the transition faults activated up to n cycles for divide-by- n clock domains unless explicitly specified using multicycle paths. Note that the tool takes into account the SDC command “create_generated_clock -divide_by” but only in the absence of the -multiply_by switch. The SDC command “create_generated_clock -multiply_by” is unsupported.

Tip

 You can use the [report_clocks](#) command to list all of the SDC-defined clocks.

- **-X_Statistics {ON | OFF}**

An optional switch and literal pair that enables the tool to collect statistics on masking that result from timing exceptions when running the [create_patterns](#) or [simulate_patterns](#) commands. When you enable this option before pattern generation or simulation, the tool collects statistics about the number of Xs captured into each to-point of timing exception paths. This option is useful to identify the timing exception paths with the most masking effect that could impact the test coverage. Collection of masking statistics is disabled by default. For an example of using this switch, refer to “[Example 7](#)” on page 1544.

- **-SLOW_Cycle_mask {ON | OFF}**

An optional switch and literal pair that specifies that the slow cycles defined in named capture procedure are treated as at-speed cycles and to apply all timing exceptions during pattern simulation. By default, the slow cycles are not considered as at-speed cycles and setup timing exceptions are not considered.

- **-ALLOW_Invalid_pin_names {ON | OFF}**

An optional switch and literal pair that specifies how the tool reports undefined SDC command objects. When set to on, the [read_sdc](#) and [add_false_paths](#) commands will report warnings when defined objects are not found. The default is off, indicating that the commands will issue error messages.

- **-MERGE_compatible_paths {ON | OFF}**

An optional switch and literal pair that specifies whether or not the tool merges compatible false paths. To enhance performance in static false path analysis and dynamic false path simulation, by default (the switch set to ON) the tool merges simple, compatible false paths into a single path.

The tool can only merge false paths if they are error free, if they have exactly the same “setup time” and/or “hold time” properties, and they meet any of these criteria:

- All the false paths that contain from-points only, and no through-points, to-points or any other conditions can be merged into a single false paths that combines all the from-points.
- All the false paths that contain to-points, and no from-points, through-points, or any other conditions, can be merged into a single false path that combines all the to-points.

- All the false paths that contain one-level through points only, and no from-points, to-points, or any other conditions can be merged into a single false path that combines all the through-points.

By default the tool merges compatible false paths and issues a summary message. For example:

```
// Note: N false paths have been merged into M paths to speed up
      timing exception analysis.
//       To disable merging for debug, you may invoke
      'set_timing_exceptions_handling -merge_compatible_paths
       off'.
```

When reporting false paths, the tool reports the merged paths and includes their original paths, as annotated with the “|+” symbol. For example:

```
// command: report_false_paths
// False path -from clk4 -through /u7 -setup (id=0)
// False path -from clk5 -through /u8 -setup (id=1)
// False Path -from clk1 clk2 clk3 -setup -hold (id=2) (merged)
// |+ False Path -from clk1 -setup -hold
// |+ False Path -from clk2 -setup -hold
// |+ False Path -from clk3 -setup -hold
```

You cannot add or delete merged paths directly. You can only add or delete the original paths and allow the tool to re-merge them. All the original false paths are preserved in the flat model. You can disable this feature by using the -off option for this switch.

Also, when you invoke switches such as “`set_timing_exceptions_handling -x_statistics on`”, the tool assumes you are debugging based upon original paths and will turn off the “`merge_compatible_paths`” feature, unless you specify it as on in the same command line.

Examples

Example 1

The following example specifies that the tool not expand timing exception paths for the clocks in asynchronous clock groups as both setup and hold timing exception paths.

`set_timing_exceptions_handling -clock_groups off`

Example 2

The following example shows how to specify reporting of SDC filename and line numbers for timing violations:

```
set_timing_exceptions_handling -line_no on
read_sdc timing.sdc
report_false_paths
```

```
// False Path -from_clock cmem_fclk1 cmem_fclk1n -to_clock p_clk
spm_pci_clk -setup (filename = \
    timing.sdc, line 406)
// False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclk1
cmem_fclk1n -setup (filename = \
    timing.sdc, line 408)
...
// Total reported false paths = 177
```

Example 3

This example shows how to specify reporting merged false paths:

```
set_timing_exceptions_handling -merge_compatible_paths on
report_false_paths -all

// Note: 2 false paths have been merged into 1
// path to speed up timing exception analysis.
// To disable merging for debug, you may invoke
// 'set_timing_exceptions_handling -merge_compatible_paths off'.

// False Path -from_clock U1/CP U2/CP -setup -hold
// False Path -from_clock U3/CP -to U8/CP -hold
// False Path -from nclk6 -setup -hold
// False Path -through G7/B G3/A nclk7 G3/A -setup -hold (merged)
// |+ False Path -through G7/B G3/A -setup -hold
// |+ False Path -through nclk7 G3/A -setup -hold
```

Related Topics

[read_sdc](#)
[report_clocks](#)
[report_false_paths](#)
[report_multicycle_paths](#)

[set_tla_loop_handling](#)

Context: all contexts

Mode: setup

Specifies how to simulate feedback loops that contain transparent latches (TLAs).

Usage

`set_tla_loop_handling [ON | OFF]`

Description

Specifies how to simulate feedback loops that contain transparent latches (TLAs).

By default, TLAs in feedback loops are simulated using the same iterative algorithm used to simulate loops through combinational logic. This can increase test coverage; however, it requires feedback loop analysis to be performed whenever DRC is run, which may result in a slightly longer DRC run time. When this command is disabled, feedback loop analysis is performed only after flattening; however, TLAs that are in feedback loops are treated as TIEX gates during simulation, which can reduce test coverage.

Note

 Be aware that extra processing time is required to support loop simulation through TLAs. In some circumstances, the improved fault detection yields sufficient reduction in ATPG effort that overall runtimes decrease, but you should anticipate an increase in runtimes in most cases.

Any TLA that is in a feedback loop is treated as a place to break the loop (similar to the FB buffer gate used for combinational loops). Strongly connected sets of gates are identified as feedback networks. You can display feedback networks by using the [report_feedback_paths](#) command.

You can use the -Environment switch of the [write_patterns](#) command to save the value of the -Iteration switch. You can use the [report_environment](#) command to see the current value of the `set_tla_loop_handling` command.

Arguments

- [ON](#)
An optional literal that enables TLA loop simulation. This is the default.
- [OFF](#)
An optional literal that disables TLA loop simulation.

Related Topics

[report_environment](#)

[report_feedback_paths](#)

[write_patterns](#)

[set_loop_handling](#)

set_tool_options

Context: all contexts

Mode: all modes

Sets certain options for Tessent Shell.

Usage

```
set_tool_options [-reapply_settings_after_reelaboration {on | off}]  
                  [-allow_vhdl_2008 {on | off}]  
                  [-instantiate_unconnected_pins_in_written_design {on | off}]
```

Description

This command sets certain options in Tessent Shell.

Arguments

- **-reapply_settings_after_reelaboration on | off**

When using [set_current_design](#) on the current design again the design will be re-elaborated.

When this setting is set to “on” Tessent Shell will preserve certain settings, if possible.

These are the settings done with the following commands:

- [add_clocks](#)
- [add_black_boxes](#)
- [add_output_masks](#)
- [add_primary_inputs](#)
- [add_primary_outputs](#)
- [add_input_constraints](#)
- [add_tied_signals](#)
- [set_output_masks](#)

Note that all settings are removed independent of this setting if you set a different current design.

- **-allow_vhdl_2008 on | off**

VHDL 2008 has a different set of IEEE STD packages compared to previous VHDL formats. You must specify to Tessent Shell if any blocks contain VHDL 2008 constructs in order to load 2008 packages. This has to be done before issuing any [read_verilog/read_vhdl](#) commands.

- **-instantiate_unconnected_pins_in_written_design on | off**

An optional switch and Boolean that specifies how the tool instantiates unconnected pins. The default is on, meaning that the tool instantiates pins even if they have no connection.

When this switch is set to off and the tool is in -no_rtl mode, the unconnected pins are not used in the following:

- Read in instances
- Created instances
- For additional pins that are created by a [replace_instances](#) command

When this switch is set to off and the tool is in -rtl mode, the unconnected pins are not used in the following:

- Created instances
- For additional pins that are created by a [replace_instances](#) command

In -rtl mode, read in instances are not changed because of format preservation.

Related Topics

[get_tool_option](#)

[set_tcl_shell_options](#)

set_trace_flat_model_options

Context: all contexts

Mode: setup, analysis

Modifies the default behavior of the trace_flat_model command.

Usage

```
set_trace_flat_model_options [-tag_condition tag_attribute_expression]
    [-stop_condition stop_attribute_expression]
    [-controllability {controlling | constant | unblocked | connected}]
    [-stop_at_first_tag {ON | OFF}] [-max_levels {unlimited | number}]
    [-latch {normal | transparent}] [-map_tag_to_design_module_boundary {ON | OFF}]
```

Description

Modifies the default behavior of the [trace_flat_model](#) command.

This command sets options as global defaults. These defaults will always be applied if they are not specified in the current trace_flat_model execution.

This command removes the requirement for you to repeatedly specify trace configuration information for each call to the trace_flat_model command.

Arguments

- **-tag_condition *tag_attribute_expression***

An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the -filter option used for the [get_pins](#) command.

- **-stop_condition *stop_attribute_expression***

An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the gate_pin failing the specified controllability requirements. The pins of memory or dff primitives are always stop points even when not explicitly specified.

- **-controllability {controlling | constant | unblocked | connected}**

An optional switch and literal pair that defines the controllability requirements before traversing a gate.

controlling—Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion,

constant—Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

unblocked—Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected—With this value, gates are always traversed when reached.

- **-stop_at_first_tag {ON | OFF}**

An optional switch and literal pair that specifies whether to interrupt tracing when the first tag object with the specified controllability is found. This is useful when checking for at least one occurrence of a given condition in the fanin or fanout of a pin without wasting time to find them all.

- **max_levels {unlimited | number}**

An optional switch and value pair that defines the maximum level of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count.

- **-latch {normal| transparent}**

An optional switch and literal pair that specifies the behavior of tracing through a latch.

normal—A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of “-controllability connected” all latch input pins - including the latch enable pin - are traced through.

transparent—All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- **-map_tag_to_design_module_boundary {ON | OFF}**

An optional switch and literal pair that specifies whether to map the tag points to gate_pins outside of library cells. When this switch is ON, the trace_flat_model command maps the tag points to gate_pins outside of library cells and never inside the cells. Library cells can be either cells from a Tessent Cell library or `celldesign Verilog modules.

Examples

The following example script shows how the set_trace_flat_model_options command uses attribute expressions to constrain the trace for level 1 clock gaters.

```
set stop_condition_expr " (isClockGater == true) || (isRegister == true) "
set tag_condition_expr "isClockGater == true"

set_trace_flat_model_options -tag_condition ${tag_condition_expr} \
    -stop_condition ${stop_condition_expr}
foreach_in_collection pin [all_clocks] {
    set CGs [trace_flat_model -from $pin -direction forward]
    if {[sizeof_collection $CGs] > 0} {
        puts "Level1 clock gaters for clock ${pin} are:"
        foreach_in_collection CG ${CGs} {
            puts " ${CG}"
        }
    }
}
}
```

Related Topics

[get_trace_flat_model_option](#)

[trace_flat_model](#)

set_trace_report

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool displays gates in the scan chain trace.

Usage

`set_trace_report OFF | ON`

Description

Specifies whether the tool displays gates in the scan chain trace.

The `set_trace_report` command controls whether the tool displays all of the gates in the scan chain trace during rules checking.

Arguments

- **OFF**
A literal that specifies for the tool not to display gates in the scan chain trace. This is the default behavior upon invocation of the tool.
- **ON**
A literal that specifies for the tool to display gates in the scan chain trace during rules checking.

Examples

Example 1

The following example displays the gates in the scan chain trace during rules checking:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Example 2

The following example displays the gates in the scan chain trace during rules checking:

```
add_clocks 0 clock
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Related Topics

[add_scan_chains](#)
[report_scan_chains](#)

[set_transcript_style](#)

Context: unspecified, all contexts

Mode: all modes

Specifies the style in which the tool transcripts commands.

Usage

```
set_transcript_style {off | full | tool_commands_only | input_only}  
    [-result_collection_limit {unlimited | limit}]  
    [-argument_collection_limit {unlimited | limit}]  
    [-one_collection_entry_per_line {on | off}]
```

Description

Specifies the style in which the tool transcripts commands.

Arguments

- **off**

A keyword that turns off all command transcription so that the commands run as though included using the Tcl “source” command. This option allows you to execute scripts that do not echo the tool commands, which can be helpful running scripts that use the report commands as required for introspection.

- **full**

A keyword that transcribes all commands read from a dofile before any Tcl evaluation is done. This includes all Tcl commands and constructs.

Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs will transcript as a single item. In addition, when non-introspection tool commands are embedded within another command, they transcript as they are about to execute. (Introspection commands are commands that simply return information about the design or tool state and therefore have no effect on operation of the tool. Tool commands included within the catch_output command are also considered introspection commands since they do not write to the transcript.)

As in a standard Tcl shell, any result returned from each top-level Tcl command transcripts when you run the tool in interactive mode. In batch mode, this option does not transcript command results.

- **tool_commands_only**

A keyword that transcribes tool commands, including all arguments, as the commands execute. The command string is transcribed after being evaluated by Tcl so all variable and command references are resolved. For commands in a loop or if/else construct, the tool transcripts each execution of each command plus any output produced so that the context is evident.

This option does not transcript Tcl flow control constructs, pure Tcl commands, and introspection commands nor their results. This option essentially specifies the legacy behavior of the tools.

- **input_only**

A keyword that transcripts all input from the dofile (both Tcl and tool commands) before sending to Tcl for evaluation. Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs transcript as a single item. Embedded tool commands are not transcribed again when they execute.

- **-result_collection_limit** unlimited | *limit*

A switch and string or integer pair that specifies number of collection entries show in the interactive mode when a collection is returned. The default is 50.

- **-argument_collection_limit** unlimited | *limit*

A switch and string or integer pair that specifies the number of collection entries show in the listing when a collection is passed to a command as an argument. The default is 1.

- **-one_collection_entry_per_line** on | off

A switch and literal pair that show the collections on arguments and results with one entry per line. The default is off.

Examples

The following example consists of a dofile and the transcription results of the various style options.

Dofile:

```
set foo 2
set myclk(1) CLK1
set myclk(2) CLK2
set pfile mytest.proc
if { $foo == 2 } {
    add_clocks 0 $myclk(1)
    add_clocks 0 $myclk(2)
}
add_scan_groups g1 $pfile
```

Full-style transcript:

```
// command: set foo 2
// command: set myclk(1) CLK1
// command: set myclk(2) CLK2
// command: set pfile mytest.proc
// command: if { $foo == 2 } {
//             add_clocks 0 $myclk(1)
//             add_clocks 0 $myclk(2)
//         }
// sub-command: add_clocks 0 CLK1
// sub-command: add_clocks 0 CLK2
// command: add_scan_groups g1 $pfile
```

Tool_commands_only-style transcript:

```
// command: add_clocks 0 CLK1
// command: add_clocks 0 CLK2
// command: add_scan_groups g1 mytest.proc
```

Input_only-style transcript:

```
// command: set foo 2
// command: set myclk(1) CLK1
// command: set myclk(2) CLK2
// command: set pfile mytest.proc
// command: if { $foo == 2 } {
//           add_clocks 0 $myclk(1)
//           add_clocks 0 $myclk(2)
//         }
// command: add_scan_groups g1 $pfile
```

Related Topics

[get_transcript_style](#)

[no_transcript](#)

[set_logfile_handling](#)

set_transient_detection

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -scan context only)

Specifies whether the tool detects all zero width events on the clock lines of state elements.

Usage

`set_transient_detection Off | {ON [-Verbose | -NOVerbose]}`

Description

Specifies whether the tool detects all zero width events on the clock lines of state elements.

The `set_transient_detection` command sets the simulator to detect all zero width events on the clock lines of state elements. If the zero width event causes a change of state in the state element, the tool sets that state element to X.

If Off is specified, DRC simulation treats all events on state elements as valid. Because the simulator is a zero-delay simulator, it is possible for DRC to simulate zero width, monostable circuits with ideal behavior which is rarely matched in silicon. The resulting zero width output pulse from the monostable circuit is also treated as a valid clocking event for other state elements.

Arguments

- **Off**

A literal that specifies for the tool to set transient detection off.

Caution

 This argument is only for special cases where you understand the potential hazards from transients on the clock lines and know how to handle them. Turning transient detection off without taking care of the hazards can result in bad patterns that mismatch when verified in a timing based simulator. Be sure you understand the consequences and are willing to accept the risks before turning transient detection off.

- **ON**

A literal that specifies for the tool to set transient detection on. This is the invocation default.

- **-Verbose**

An optional switch that specifies for the tool to display a message identifying each time a state element is set to X due to transient detection. This is the invocation default and is valid only when transient detection is enabled.

- **-NOVerbose**

An optional switch which specifies for the tool *not* to display a message identifying each time a state element is set to X due to transient detection. This switch is valid only when transient detection is enabled.

Examples

Transient detection is enabled in verbose mode by default. Unless you use the **set_transient_detection** command to modify the default, you will see a message similar to the following for each transition that is caused by a transient, or zero-width, pulse:

```
// Warning: 0 width pulse detected on pin 2 of  
/my_design/my_inst/my_latch/ (463228) in procedure my_procedure at time 0.
```

You can eliminate these messages, while retaining the advantages of transient detection, by issuing this command:

set_transient_detection on -noverbose

Or you can disable transient detection, causing DRC simulation to treat all events on state elements as valid.

set_transient_detection off

set_transition_holdpi

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.

Usage

set_transition_holdpi ON | OFF

Description

Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.

The **set_transition_holdpi** command lets you turn this feature on while the fault type is in “transition”. This is useful in cases where it is not practical to maintain high data rates to the primary input pins of the device under test on a tester. Primary inputs may still change from pattern to pattern however.

Note

 Because launch off shift transition patterns require the scan enable signal (which is not a clock or RAM control) to transition within a single pattern, the tools ignore this command when generating this type of pattern.

Arguments

- **ON**

A literal that specifies for the tool to hold all primary input values (other than clocks and RAM controls) during multiple cycles of pattern generation.

- **OFF**

A literal that specifies for the tool not to hold all primary input values. This is the invocation default.

Examples

The following example holds all primary input values.

```
set_transition_holdpi on
```

Related Topics

[report_primary_inputs](#)

[set_pathdelay_holdpi](#)

[set_tristate_gating](#)

Context: dft -scan, dft -test_points

Mode: setup

Specifies how tri-state devices are controlled during scan chain shifting.

Usage

```
set_tristate_gating {OFF | ON | Busdrivers | Scan | primary_input_or_output...}  
[-Control {SEN | TEn}] [-Force_gating]
```

Description

Specifies how tri-state devices are controlled during scan chain shifting.

The set_tristate_gating command ensures tri-state devices are controlled to either prevent bus contention or be turned on during testing. If the necessary test logic already exists, none is inserted.

When enabled, test logic is inserted that controls tri-state devices as follows:

- **Multiple tri-state gates driving nets (bus net)** — One gate is turned on and the rest are turned off during testing.
- **Single tri-state gates driving nets (primary net or an internal net)** — All single tri-state gates are turned on for testing.

You can also specify which signal (SEN or TEN) controls the enable lines of tri-state devices.

By default, when the enable signal of a tri-state device is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

Arguments

- {**OFF** | **ON**| **Busdrivers**| **Scan**| *primary_input_or_output...*}

Required literal or repeatable string that specifies which tri-state devices to control during scan shifting. The default setting is off. Options include:

OFF — no test logic is inserted to control tri-state devices. Default setting.

ON — test logic is inserted when necessary to control all tri-state devices.

Busdrivers — test logic is inserted to control tri-state devices driving bus nets.

Scan — test logic is inserted to control tri-state devices used as scan inputs/outputs.

primary_input_or_output — specifies a primary input or output pin. Test logic is inserted to control the tri-state devices driving the specified primary output pin(s) or driven by the specified primary input pin(s).

- -Control SEn | TEn

An optional switch and literal pair that specifies the enable signal used to control tri-state devices. Literal options include:

SEn — Specifies the scan_enable signal. Default setting.

TEn — Specifies the test_enable signal.

- -Force_gating

An optional switch that adds test logic to the enable lines of tri-states devices when these lines are directly controlled by primary inputs, or by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Examples

Example 1

The following example uses the set_tristate_gating command to insert test logic and make all tri-state devices driving bus nets controllable via the SEN signal.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
set_system_mode analysis
add_scan_mode unwrapped -si_connections c1 bidi_in1/X \
    -so_connections blkB1/blkA/utri2/A
analyze_scan_chains
```

Related Topics

[report_test_logic](#)

[report_control_signals](#)

[set_bidi_gating](#)

[set_test_logic](#)

set_tsdb_output_directory

Context: unspecified, dft, patterns

Mode: setup

Sets the TSDB output directory used by the process_dft_specification command.

Usage

```
set_tsdb_output_directory directory_path
```

Description

Sets the TSDB output directory used by the [process_dft_specification](#) command.

The TSDB output directory defaults to `./tsdb_outdir` until the `set_tsdb_output_directory` command has been invoked. When either the `get_tsdb_output_directory` or `set_tsdb_output_directory` command is invoked, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it does not.

The `process_dft_specification` command is currently the only command that makes use of the TSDB directory.

Arguments

- *directory_path*

A required string value used to specify the TSDB output directory which is populated by commands such as `process_dft_specification`. The specified `directory_path` can be a relative path with respect to the current working directory or an absolute path. In either case, the directory must be a writable directory when it exists, or the effective parent directory where it is to be created must be a writable directory. The `set_tsdb_output_directory` command does not create the directory; it is created the first time a command needs to store information into it such as when the `process_dft_specification` command is invoked.

Examples

Example 1

The following example set the TSDB output directory to `“./dir1”`.

```
set_tsdb_output_directory dir1
```

Example 2

The following example shows a possible error than can be generated when setting the TSDB output directory.

```
set_tsdb_output_directory dir2/dir1
```

```
// Error: The parent directory of the specified tsdb_output_directory '/  
home/user1/dir2' does not exist.
```

Related Topics

[process_dft_specification](#)
[get_tsdb_output_directory](#)

[set_visualizer_logging](#)

Context: unspecified, all contexts

Mode: setup, analysis

Writes the commands entered into the Transcript window to the file specified by the *enhanced_dofile* argument.

Usage

```
set_visualizer_logging {{-STArt enhanced_dofile [-Replace | -Append]} | -STOp}
```

Description

Writes the commands entered into the Transcript window to the file specified by the *enhanced_dofile* argument.

Because only executable commands are written to *enhanced_dofile*, you can use the file like a normal dofile to recreate a previous view in DFTVisualizer.

Arguments

- **-Start**

A required argument that specifies to begin writing each command that is entered into the Transcript window to the *enhanced_dofile* file.

- ***enhanced_dofile***

A required string that specifies the name of the file into which you want to write the commands entered into the Transcript window. Only executable commands are written to this file.

- **-Replace**

An optional switch that specifies to replace the contents of *enhanced_dofile* if that file already exists.

- **-Append**

An optional argument that specifies to add the output of the logging operation to the *enhanced_dofile* file.

- **-Stop**

A required argument that specifies to stop writing commands to the *enhanced_dofile* file.

Examples

The following example unselects an object identified by its instance name:

```
set_visualizer_logging -start my_logfile
```

Related Topics

[open_visualizer](#)

set_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Controls a subset of DFTVisualizer preferences for the Flat Schematic and Hierarchical Schematic windows.

Usage

```
set_visualizer_preferences [-Compact | -NOCompact] [-NOQuery  
| -Query threshold_number] [-Split | -NOSplit] [-Display {FLAt_schematic |  
HIEarchical_schematic | ALL}]
```

Description

Controls a subset of DFTVisualizer preferences for the Flat and Hierarchical Schematic windows.

This command controls the following three aspects of DFTVisualizer schematic display behavior:

- Display of gates
- Whether the tool will query you for confirmation before beginning to process data to display a large schematic
- Display of schematics in multiple sheets

There are invocation defaults for all the settings. When you use this command to change any of the settings, the specified preferences (along with any window displays affected by the settings) are updated immediately.

Arguments

- -Compact

A switch that specifies to display only gates that could have a logical impact on the output results. This is the invocation default behavior.

This switch is supported in the Flat Schematic window only.

- -NOCompact

A switch that specifies to display all netlist gates, including buffers, inverters, ZVAL, and single-input bus gates. Many of these types of gates may not affect the logical output results, yet tend to clutter the display. The clutter is most noticeable when the gate level is set to primitive (set_gate_level Primitive).

This switch is supported in the Flat Schematic window only.

- **-NOQuery**

A switch that specifies for DFTVisualizer to display any netlist you request, regardless of its gate count.

- **-Query *threshold***

A switch and integer pair that specifies the maximum number of gates that may be added to the Flat and Hierarchical Schematic windows without first asking if you want to continue. Displaying netlist segments containing a relatively large number of gates can increase run time. This switch enables you to set a gate count threshold at which the tool will prompt you for confirmation that you do want to add more than the *threshold* number of gates. The default threshold is 500 gates.

- **-Split**

A switch that specifies to split the schematic into multiple schematic sheets and display just one sheet at a time. This is the invocation default behavior.

In -Split mode, an off-sheet connector on a net indicates a connection to a gate on another sheet. Double click on the off-sheet connector to change the display to the other sheet. At the bottom of the display, to the right of the message area, the tool lists the sheet currently being displayed and the total number of sheets. Entering a sheet number in the sheet number entry box is another way to change the display to another sheet.

This switch is supported in the Flat Schematic window only.

- **-NOSplit**

A switch that specifies for DFTVisualizer not to split the schematic into multiple schematic sheets.

This switch is supported in the Flat Schematic window only.

- **-Display {FLAt_schematic| HIErarchical_schematic | ALL}**

A switch and literal that specify the window(s) to which this command's settings apply. When you use the -Display switch, you have the following three choices:

FLAt_schematic — A literal that applies the settings to the Flat Schematic window only.
This is the invocation default behavior.

HIErarchical_schematic — A literal that applies the settings to the Hierarchical Schematic window only.

ALL — A literal that applies the settings to both the Debug and Design Flat and Hierarchical Schematic windows.

Examples

The following example changes the setting of the query threshold for both the Flat and Hierarchical Schematic windows. The compaction and sheet splitting options continue to use the default settings.

```
set_visualizer_preferences -query 200 -display all
```

Related Topics

[add_display_instances](#)
[analyze_drcViolation](#)
[open_visualizer](#)
[report_gates](#)
[set_gate_report](#)

set_wrapper_analysis_options

Context: dft -scan

Mode: setup, analysis

This command sets up parameters for the wrapper cells analysis.

Usage

```
set_wrapper_analysis_options
  [-output_fanin_flop_threshold {32 | integer | unlimited}]
  [-input_fanout_flop_threshold {32 | integer | unlimited}]
  [-output_fanin_libcell_levels_threshold {32 | integer | unlimited}]
  [-input_fanout_libcell_levels_threshold {32 | integer | unlimited}]
  [-non_scan_depth_threshold {2 | unlimited | integer}]
  [-register_ports_exceeding_non_scan_depth on | off]
  [-register_undriven_output on | off]
  [-allow_internal_segments_as_wrapper on | off]
  [-pipeline_start_of_wrapper_chains on | off]
  [-dedicated_wrapper_cells_location inside_power_isolation | outside_power_isolation]
  [-exclude_ports port_spec]
  [-clear_excluded_ports]
  [-register_ports_reaching_blackboxes {blackbox_instances_list | all | off}]
```

Description

This command sets up parameters for the wrapper cells analysis.

Arguments

- `-output_fanin_flop_threshold {32 | integer | unlimited}`

An optional switch and integer or literal pair that specifies the maximum number of sequential elements allowed to be reached during the backward tracing from a PO. The default is 32.

For the scan segments (sub-chains) encountered during the backward tracing from POs, there is a special cost function to average the number of the backward hit flops between the total number of POs reaching this sub-chain and the length of this sub-chain.

- `-input_fanout_flop_threshold {32 | integer | unlimited}`

An optional switch and integer or literal pair that specifies the maximum number of sequential elements allowed to be reached during the forward tracing from a PI. The default is 32.

For the scan segments (sub-chains) encountered during the forward tracing from PIs, there is a special cost function to average the number of the forward hit and the backward flops (feedback flops) between the total number of PIs reaching this sub-chain and the length of this sub-chain.

- **-output_fanin_libcell_levels_threshold {32 | integer | unlimited}**

An optional switch and integer or literal pair that specifies the maximum number of levels of library cells to be traversed during the backward tracing from a PO until the first sequential element is reached. The default is 32.

- **-input_fanout_libcell_levels_threshold {32 | integer | unlimited}**

An optional switch and integer or literal pair that specifies the maximum number of levels of library cells to be traversed during the forward tracing from a PI until the first sequential element is reached. The default is 32.

- **-non_scan_depth_threshold {2 | integer}**

An optional switch and integer or literal pair that controls the depth of non_scan sequential elements traced by the tool. When the threshold is exceeded, the behavior depends on the **-register_ports_exceeding_non_scan_depth** switch. The default is 2.

- **-register_ports_exceeding_non_scan_depth on | off**

An optional switch and literal pair that controls the behavior of the tool when the non-scan depth threshold is exceeded. If it is set to on, a dedicated wrapper cell will be added to the port that is being traced. If it is off, the branch that exceeds the threshold will be ignored and the tracing will continue to search for shared wrapper cell candidates.

- **-register_undriven_output on | off**

An optional switch and literal pair that specifies whether an output port having no functional source should receive a dedicated wrapper cell.

- on — undriven outputs will receive dedicated wrapper cells except for specific ports that are configured with “**set_dedicated_wrapper_cell_options off –ports {...}**”.
- off — undriven output ports will not receive any wrapper cells except for specific ports that are configured with “**set_dedicated_wrapper_cell_options on –ports {...}**”. This is the default.

- **-allow_internal_segments_as_wrapper on | off**

An optional switch and literal pair that specifies whether to identify the defined sub-chains as Input or Output wrapper elements during the forward and backward tracing from the Primary IOs.

- on — identify encountered sub-chains as Input or Output wrapper elements. This is the default.
- off — do not identify encountered sub-chains as wrapper elements. Ports which reach sub-chains will have a dedicated wrapper cell inferred unless these ports are specified to not allow inferring dedicated wrapper cells for them (see **set_dedicated_wrapper_cell_options**)

- **-pipeline_start_of_wrapper_chains on | off**

An optional switch and literal pair that specifies whether to automatically insert an at-speed test flop at the beginning of each wrapper chain. When multiple clock domains are allowed

per chain, the tool breaks the scan path with SE and adds a pipelining at the beginning of each domain so as not to create domain interaction through the scan path. The default is on.

- **-dedicated_wrapper_cells_location inside_power_isolation | outside_power_isolation**
An optional switch and literal pair that specifies where to insert the Input and Output dedicated wrapper cells with relation to the encountered power isolation cells.
 - inside_power_isolation — insert Input dedicated wrapper cells after the encountered power isolation cells and the Output dedicated wrapper cells before the encountered power isolation cells. This is the default.
 - outside_power_isolation — insert Input wrapper cells immediately after the PI ports being registered and insert the dedicated Output wrapper cells immediately before the PO ports being registered.
- **-exclude_ports *port_spec***
An optional switch and string that specifies that all ports in *port_spec* will be excluded from wrapper analysis without regard to their previous state (off/on/auto). Ports can only be removed from exclusion with **-clear_excluded_ports** switch.
port_spec can be a Tcl list of one or more ports or a collection of one or more port objects.

If one or more ports in the port spec has already been set with
“**set_dedicated_wrapper_cell_options on|off –ports {...}**” a note will alert you that this setting has been overridden.

```
// Note: Excluding and overriding set_dedicated_wrapper_cell
configurations for the following port(s):
<port1>, <port2>, <port3> ...
```

- **-clear_excluded_ports**
An optional switch that clears the list of ports that have been excluded from wrapper analysis. These ports are now included in wrapper analysis and will receive shared, dedicated, or no wrapper cells based on future and/or global “**set_dedicated_wrapper_cell_options [on|off|auto]**” configurations.
- **-register_ports_reaching_blackboxes {*blackbox_instances_list* | all | off}**
An optional switch and a list of blackbox instances or a literal that directs the tool to add a dedicated wapper cell to ports connected to the specified blackbox instances. The argument choices are as follows:
 - *blackbox_instances_list* — Adds dedicated wrapper cells to ports connected to any blackbox instance in the list.
 - all — Adds dedicated wrapper cells to ports connected to any blackbox instance.
 - off — A literal that disables this feature. This is the default.

Note

 This command switch is cumulative, and every invocation adds to the list of blackbox instances. Using the off literal clears the list.

Examples

Example 1

In the following example, all Primary IOs except for excluded_1 will receive dedicated wrapper cells, including undriven IOs.

```
set_wrapper_analysis_options --exclude_ports excluded_1
set_dedicated_wrapper_cells on
set_wrapper_analysis_options --register_undriven_output on
```

Example 2

In the following example:

Ports A, B, C, D, excluded_1 are driven.

Ports undriven_1, undriven_2 are undriven.

Ports C, D, undriven_1 will receive dedicated wrapper cells

Ports A, B, undriven_2, excluded_1 will not receive dedicated wrapper cells.

```
set_wrapper_analysis_options --exclude_ports excluded_1
set_wrapper_analysis_options --register_undriven_output on
set_dedicated_wrapper_cells off --ports {A B undriven_2}
```

Related Topics

[set_dedicated_wrapper_cell_options](#)

[set_write_patterns_options](#)

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis, setup

Creates the subset of ports or vector callbacks to use with the write_patterns command.

Usage

Usage for Port Lists

```
set_write_patterns_options [-additional_port_list add_port_name]  
[-existing_used_ports used_port_name] [-reset_used_ports]
```

Usage for Vector Callbacks

```
set_write_patterns_options [-vector_callback vector_callback]  
[-received_ports received_ports] [-returned_ports returned_ports] [ -replace ] ]  
[-delete_vector_callbacks]
```

Description

Creates the subset of ports or vector callbacks to use with the [write_patterns](#) commands. See “[Vector Creation and Modification](#)” in the *Tessent Scan and ATPG User's Manual* for additional information.

Port List Usage

This command is used to create the subset of ports to use in the subsequent write_patterns commands.

The add_port_name argument is a tcl list of object names or a collection of objects. The valid object types are ports , pseudo-port, pin, or gate_pin. A pin must map to a pseudo-port and a gate pin must map to a pseudo-port or port. The action is to add port names to the used subset, assuming that we are starting from an empty subset. Multiple instances of this command are not cumulative. Each command first removes any existing subset, and starts over from the full port list. Issuing the –existing_used_ports option with an empty *obj_spec* such as {} creates an empty subset. To add ports to the port list that do not exist in the original design data, use the –additional_port switch to specify a list of lists of strings which specify the name to add and its port direction. Port directions can be one of input, output, or inout. Optionally, the clock attribute and an optional off state for a clock can also be specified. If no off state is specified, an off state of 0 is assumed. If “clock” is not specified, the additional port is not treated as a clock and the P state value cannot be assigned to it. If “clock” is specified, the additional port is treated as a clock, the P state value can be used, and the timing for the port is taken from the “pulse_clock” statement in the timeplate instead of “force_pi” statement. Issuing the command with only –additional_ports will only add ports to the default port list supplied by the design and setup information. If any additional ports are added that match the name of an existing port, an error will be issued. This example instructs the write_patterns commands to only include the ports starting with ijtag_ and two new ports called myNewInput and myNewOutput[1:0].

```
set_write_pattern_options -existing_used_ports [get_ports iJtag_*] \
    -additional_port_list {[list myNewInput input myNewOutput[1:0] output]}
```

Vector Callback Usage

This command adds a vector callback proc name to the list of vector callback procs that will be used in the subsequent write_patterns commands. The switch –vector_callback must specify the name of a defined tcl proc that will be passed a vector set and will return a vector set. An optional trigger can be specified as an activation criterion for this Tcl proc. If no trigger is specified, the every_cycle trigger is the default and this callback is used for all vectors. Only one callback can be specified with each trigger. If another callback is specified with the same trigger as a previously specified callback, this new callback replaces the older one if -replace is used otherwise an error is issued. If a new callback is added with the same name as a previous one, it replaces the previous one if –replace is used, otherwise an error is issued.

When using vector_callback procs and writing patterns, the data fed to the vector callback procs must be the internal view pattern data, and therefore, any external_mode cycles from a named capture, or external_mode cycles from PLL_cycles or external_capture are not used. When you enable vector callbacks, the tool issues the following warnings:

```
// Warning: The set_write_pattern_options -vector_callback command causes
// write_patterns to use the internal_mode view of the pattern data.
// Set_external_capture_options -PLL_cycles is ignored in that case.
```

Or:

```
// Warning: The set_write_pattern_options -vector_callback command causes
// write_patterns to use the internal_mode view of the pattern data.
// Set_external_capture_options -capture_proc is ignored in that case.
```

Arguments

- **-additional_port_list *add_port_name***

An optional switch and value pair that specifies a list of lists of strings which specify the name to add and its port direction using the following syntax:

```
-additional_port_list {{port_name input | output | inout [clock <0|1>]} ...}
```

- **-existing_used_ports *used_port_name***

An optional switch and value pair that specifies an absolute list of the subset of existing design ports to use when writing patterns. Refer to “[Controlling Ports Used in Pattern and Testbench Generation](#)” on page 1182 in the [process_patterns_specification](#) command description for additional information and usage notes.

- **-reset_used_ports**

An optional switch used to reset the port list back to the original port list supplied by the design and setup information contained in the tool.

- **-vector_callback** *vector_callback* [-received_ports *received_ports*] [-returned_ports *returned_ports*] [-replace]

An optional switch and value pair that adds a vector callback proc name to the list of vector callback procs that will be used in subsequent write_patterns commands.

-received_ports *received_ports* — Specifies a Tcl list of port names or gate_id or a collection of ports or gate that is used as the callback port list when passing a vector set to the callback.

-returned_ports *returned_ports* — Specifies a Tcl list of port names or gate_id or a collection of ports or gate that is used as the callback port list when returning a vector set from the callback.

- **-delete_vector_callbacks**

An optional switch that removes all existing vector callbacks.

Related Topics

[get_write_patterns_options](#)

[report_write_patterns_options](#)

[write_patterns](#)

[Vector Creation and Modification](#)

set_xbounding_options

Context: dft -scan, dft -test_points

Mode: setup

Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.

Usage

```
set_xbounding_options [-xbounding_enable xbounding_enable_name]
                      [-connect_to {existing_scan_cell | new_scan_cell | constant_value}]
                      [-scan_drive_limit integer]
                      [-mcp_bounding_enable mcp_enable_name]
                      [-exclude_sdc_cross_domain_paths {off | on}]
                      [-test_clock {off | on}] [-exclude pin_or_port]
                      [-use_scan_cells {off | on}] [-bound_clock_gater_enables {on | off}]
```

Description

Sets up the parameters for X bounding insertion. The analysis is triggered by the [analyze_xbounding](#) command.

You use this command to set various options that control how the tool performs X-bounding analysis. These options include:

- The name of the enable signals that activate the bounding logic.
- The type of bounding (new scan cells, existing scan cells, or constant values) that is used to block the X source.
- Activate (or disable) bounding of cross-clock domain paths as specified by the SDC file.

The actual analysis must be triggered by issuing the [analyze_xbounding](#) command.

If you issue this command without any arguments, all the defined and default settings are reported.

Arguments

- **-xbounding_enable *xbounding_enable_name***
An optional switch and literal pair used to specify the name of the of the X-bounding enable signal.
- **-connect_to {existing_scan_cell | new_scan_cell | constant_value}**
An optional switch and literal pair used to specify whether the X sources are controlled by existing or new scan cells or whether the X source is blocked with a constant value.
Choose from one of the following:
 - existing_scan_cell* — Specifies using existing scan cells to control the X sources.

new_scan_cell — Specifies using new scan cells to control the X sources. When using new scan cells for X bounding, the scan cell model specified either by the [add_cell_models](#) command or the **cell_type** Tessent Cell library attribute is used.

constant_value — Specifies using either an AND gate to block the X source with a constant 0 or an OR gate to block the X source with a constant 1. The specific bounding value (0 or 1) is determined by performing a simple analysis of the logic that will be forced to the constant value.

- **-scan_drive_limit *integer***

An optional switch and integer pair that specifies a limit on how many Mux-type control points one scan cell can drive for bounding. Having one scan cell drive a large number of Mux control points can reduce test coverage, due to structural dependency. 0 means there is no limit. The invocation default is 1.

- **-mcp_bounding_enable *mcp_enable_name***

An optional switch and string pair that specifies the name of an alternate enable signal for bounding at-speed paths. This provides independent control of these bounding muxes as the tool must bound multicycle paths during at-speed testing.

- **-exclude_sdc_cross_domain_paths {off | on}**

An optional switch and literal that specifies whether the tool bounds cross clock-domain paths that are specified in the SDC file.

A cross clock-domain path occurs when memory elements driven by one clock domain, feed logic that is observed (captured) by memory elements that are operated by a different clock.

Cross clock-domain paths are typically specified as false paths in the SDC file. In that case, these paths are identified as potential X sources and bounding muxes are inserted near the capture point.

In some cases, this might constrain the clock sequences that are applied during Logic BIST such that these paths will never be sensitized. However, if your hardware is designed such that certain cross clock_domain interactions are not tested during Logic BIST, and you choose to not test these the cross clock_domain paths by constraining the capture sequences, then there is no need to insert bounding muxes.

The **-exclude_sdc_cross_domain_paths** switch instructs the tool to ignore false paths that are specified in the SDC file during analysis.

Note

 In order to guarantee that all cross clock domains are targeted for x-bounding, you must specify all cross clock domain paths as false paths in the SDC file. This can be done indirectly by using the **set_clock_groups** command to assign each clock to a separate group.

- **-test_clock {off|on}**

An optional switch and literal pair that determines the clock signal for the newly inserted scan cells that will drive the test_mode input of the x-bounding muxes. This option must be used in conjunction with the -connect_to new_scan_cell option.

off — A literal that specifies that the clock signal for the newly created scan cells will be chosen to match the clock domain of the flops in the fanout cone of the x-bounding mux.

on — A literal that specifies that the newly created scan cells will be driven by the test clock signal as defined with the “set_scan_signals -tclk” command.

- **-exclude *pin_or_port***

An optional switch and string pair that specifies the pins/ports that are guaranteed to have a known value during fault simulation and will never propagate an X value to the MISR.

The value of *pin_or_port* can be a Tcl list of one or more pins/ports, or a collection of one or more pin/port objects.

Note

 The tool will not insert X-bounding muxes at the specified pins/ports, or at the logic that is driven by these pins. During fault simulation the X-sources will not be ignored, which means you must manually apply the appropriate initialization and constraints in order to avoid X's reaching the MISR during fault simulation.

- **-use_scan_cells {off|on}**

An optional switch and literal pair that controls the tool behavior for the following specific scenario:

- x-bounding is done on a pre-scan netlist.
- the “-connect_to new_scan_cells” option was used.
- you do not issue the “analyze_scan_chains” command after analyze_xbounding.

When this switch is off (this is the default) the tool will insert non-scan flops to drive the test_mode input of the x-bounding muxes.

When the switch is on, the tool will insert scan_cells. However, since analyze_scan_chains was not called, these new scan flops will NOT be stitched into chains.

- **-bound_clock_gater_enables {on|off}**

An optional switch and literal pair that controls x-bounding of false or multi-cycle paths that reach the functional enable input of clock gaters that drive scannable cells.

When set to on (this is the default), the tool will add x-bounding logic at the test_enable input of the clock gater to force it to always be enabled such that the potential x at the functional enable will not have any impact.

When set to off, the tool does not perform any bounding of such paths.

Examples

Example 1

In the following example, when you issue this command without any arguments, it reports the current settings:

set_xbounding_options

```
Xbounding enable name      : lbist_en
Connect to                 : existing scan cell
Scan drive limit          : 1
Exclude sdc cross domain paths : OFF
MCP bounding enable name   : lbist_en
Use test clock             : OFF
Exclude pin or port list  : DataOut_i[0]
                           : DataOut_i[1]
                           : DataOut_i[2]
```

Example 2

The following command specifies non-default enable names for the primary enable signal as well as the false and multicycle path enable prior to performing the x-bounding analysis:

```
set_xbounding_options -xbounding_enable lbist_en \
-mcp_bounding_enable mcp_path_enable
analyze_xbounding
```

Related Topics

[analyze_xbounding](#)
[insert_test_logic](#)
[report_xbounding](#)

set_xclock_handling

Context: all contexts

Mode: setup, analysis

Specifies whether the sequential element model outputs X when any of its clock inputs become X.

Usage

```
set_xclock_handling {Retain | X} [-Pessimistic_simulation {ON | OFF}]
```

Description

Specifies whether the sequential element model outputs X when any of its clock inputs become X.

By default, the flip-flop and latch default models are used for sequential elements. These default models retain their output values when their clock value becomes X so long as the other input values do not cause the stored output value to change, regardless of whether the clock was at a 0 or 1. The set_xclock_handling command lets you override this behavior during pattern simulation, causing the stored output value to become X.

[Table 6-21](#) lists the Q capture value of the level-sensitive sequential element models based on the X values on input pins, the Q value, and on the set_xclock_handling arguments. [Table 6-22](#) lists the outputs of edge-triggered sequential element models based on the clock transition, the D value, and on the set_xclock_handling arguments.

This command has no effect during DRC simulation. The default flip-flop and latch behavior is used during DRC simulation.

Table 6-21. X Handling for Level-Sensitive Ports

Level-Sensitive Sequential Element Ports		Q capture value when using the Retain argument	Q capture value when using the X argument
latch enable = X	Q=D	Q	X
	Q≠D	X	X
set = X	Q=0	X	X
	Q=1	1	X
reset = X	Q=0	0	X
	Q=1	X	X

Table 6-22. X Handling for Edge-Triggered Ports

Edge-triggered Sequential Element Inputs		Output when using “Retain -pessimistic_simulation OFF / ON”		Output when using “X -pessimistic_simulation OFF / ON”	
Clock event	D value before clock event	OFF	ON	OFF	ON
X→X X→1 0→X	D=Q	Q	Q	X	X
	D≠Q	X	X	X	X
1→X	D=Q	Q	Q	Q	X
	D≠Q	Q	X	Q	X
X→0	D=Q	Q	Q	Q	Q
	D≠Q	Q	Q	Q	Q

Arguments

- **Retain | X**
A required literal that uses the default sequential element model (by default) or uses a modified sequential element model. For information about the effect of using these arguments, refer to [Table 6-21](#) and [Table 6-22](#).
- **-Pessimistic_simulation {ON | OFF}**
An optional switch that affects the output of the edge-triggered sequential element model when the clock port becomes X, as shown in [Table 6-22](#).

Examples

The following example sets the outputs of edge-triggered sequential elements to X unless any of their clock inputs transition from X to 0 during pattern simulation:

```
set_xclock_handling x -pessimistic_simulation on
```

Related Topics

[get_xclock_handling](#)

set_z_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies the simulation handling for high impedance signals on internal and external tri-state nets.

Usage

set_z_handling {Internal state} | {External state}

Description

Specifies the simulation handling for high impedance signals on internal and external tri-state nets.

The set_z_handling command specifies how to handle the high impedance state for internal and external tri-state nets during pattern simulation. If the high impedance value can be made to behave as a binary value, certain faults may become detectable. If you do not use this command to set the Z handling, the default value is X.

This command does not impact expected values on tri-state and bidirectional pins in the final saved patterns. The command changes the behavior of how Z values are simulated on internal nets. By default, Z values are treated as X values when fed into another internal gate, such as an AND gate. However, a Z value on a tri-state output pin is not converted to an X by default.

Arguments

- **Internal state**

A literal pair that specifies how the tool simulates high impedance values for internal tri-state nets. The literal choices for the *state* option are as follows:

X — A literal that specifies to treat high impedance states as an unknown state. This is the default behavior upon invocation of the tool.

0 — A literal that specifies to treat high impedance states as a 0 state.

1 — A literal that specifies to treat high impedance states as a 1 state.

- **External state**

A literal pair that specifies how the tool simulates high impedance values for external tri-state nets. The literal choices for the *state* option are as follows:

X — A literal that specifies not to measure high impedance states; they cannot be distinguished from a 0 or 1 state. This is the default behavior upon invocation of the tool.

0 — A literal that specifies to treat high impedance states as a 0 state that can be distinguished from a 1 state.

- 1** — A literal that specifies to treat high impedance states as a 1 state that can be distinguished from a 0 state.
- Z** — A literal that specifies to uniquely measure the high impedance states; they can be distinguished from both a 0 or 1 state.

Examples

The following example specifies to treat high impedance values as 1 states when they feed into logic gates, and as 0 states at the output of the circuit, then performs a pattern simulation run:

```
set_z_handling internal 1
set_z_handling external 0
set_system_mode analysis
create_patterns
```

Related Topics

[write_patterns](#)

[set_zhold_behavior](#)

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether ZHOLD gates retain their state values.

Usage

`set_zhold_behavior OFF | ON`

Description

Specifies whether ZHOLD gates retain their state values.

The ZHOLD gate allows the tool to model tri-state nets so that they can retain the previous state value when the net goes to a high impedance (Z) value. ZHOLD gates normally require clock sequential patterns to utilize this capability. But, if a ZHOLD gate is set to a fixed binary value when the clocks are off, the ZHOLD gate can retain that value for combinational patterns. The combinational fault simulation does not consider the fault effect on the retained value.

By invocation default, the tool assumes the retained value is always the `clock off_state` value for both the good machine and the faulty machine simulation.

The tool cannot assume a state from a previous scan pattern or from a load operation. The amount of time the ZHOLD gate can hold its value is limited to the number of sequential clock cycles.

Arguments

- **OFF**

A literal that specifies not to allow ZHOLD gates to retain values.

- **ON**

A literal that specifies for the tool to use ZHOLD gates to retain values subject to restrictions that the rules checker identified. This is the default behavior upon invocation of the tool.

Examples

The following example specifies that the ZHOLD gates are not allowed to retain their previous values:

```
set_zhold_behavior off
```

Related Topics

[report_gates](#)

[set_learn_report](#)

setenv

Context: unspecified, all contexts

Mode: all modes

Sets a shell environment variable within the tool environment.

Usage

`setenv variable_name [value]`

Description

Sets a shell environment variable within the tool environment.

The setenv command sets or changes the value of a shell environment variable within the current tool environment, enabling the tool to access the variable for the rest of the session just as if the variable had been set from the shell. This command is useful if you ever find that the tool requires an environment variable that was not set prior to invocation.

Note

 A shell variable you set with this command is set only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To unset the variable before you exit, use the [unsetenv](#) command. To view the value of the variable, use the command to pass an appropriate shell command ([echo](#) or [env](#) for example) to the operating system for execution.

Arguments

- ***variable_name***
A required string that specifies the name of the variable.
- ***value***
An optional string that specifies the value of the variable.

Examples

The following example assigns a value to the environment variable `SSH_AGENT_PID`, then displays its value:

```
setenv SSH_AGENT_PID 66216
system {echo $SSH_AGENT_PID}
66216
```

Related Topics

[unsetenv](#)

[system](#)

shutdown_sid_tester

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Terminates the currently active SID tester process that was created by executing the [launch_sid_tester](#) command.

Usage

`shutdown_sid_tester`

Description

Terminates the currently active SID tester process that was created by executing the [launch_sid_tester](#) command.

Arguments

None

Examples

The following example shuts down the SID tester process.

`shutdown_sid_tester`

Related Topics

[launch_sid_tester](#)

[simulate_clock_pulses](#)

Context: all contexts

Mode: setup, analysis

Prerequisites: The flat model must already exist, and you must have previously used the add_clocks command to define specified gate_pin objects as clocks.

Pulses one or more clocks within the current simulation context.

Usage

`simulate_clock_pulses obj_spec [-repetitions number]`

Description

Pulses one or more clocks within the current simulation context.

The simulate_clock_pulses command pulses the specified clock(s) within the current simulation context. Before issuing this command, you must previously use the add_clocks command to define the specified gate_pin objects as clocks.

The default number of clock pulses is 1 unless you specify a different number with the -repetitions switch. If any forces have been queued with previous add_simulation_forces commands, the tool automatically simulates these forces before pulsing the clocks.

Arguments

- *obj_spec*
A required value that specifies one or more gate_pin objects.
- -repetitions *number*
An optional switch and integer pair that specifies the number of clock pulses to simulate. If you do not use this switch, the default number of clock pulses is 1.

Examples

The following example shows the use of the simulate_clock_pulses command:

```
SETUP> add_clocks myInst1/OUT
SETUP> add_clocks myInst2/OUT -pseudo_port_name myClk
SETUP> simulate_clock_pulses [get_gate_pins {myInst1/OUT myClk} ] -repetitions 2
```

Related Topics

- [add_simulation_context](#)
- [get_simulation_context_list](#)
- [add_simulation_forces](#)
- [get_simulation_value_list](#)
- [copy_simulation_context](#)

[report_simulation_contexts](#)
[delete_simulation_contexts](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[simulate_forces](#)

[simulate_forces](#)

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Simulates the queued forces in the current simulation context.

Usage

```
simulate_forces [-latch {normal | transparent}]
```

Description

Simulates the queued forces in the current simulation context.

The simulate_forces command simulates all the queued simulation forces previously added with the add_simulation_forces command. This command operates only within the current simulation context.

Arguments

- **-latch {normal | transparent}**

An optional switch and value pair that controls how latches are to be handled when propagating simulation values.

A value on the D input of a latch will propagate to the Q output of the latch when the enable pin of the latch is simulated to its active value. When the enable pins of the latch is simulated to its inactive value, the Q pin retains its value. The -latch switch controls the behavior when the enable pin of the latch simulates to an X.

Choose one from the following literals:

normal — The output of the latch will go X if the D and Q values are different.

transparent — The output of the latch will be updated to match the value on the D pin.

This switch is needed when you are trying to extract clock paths from the destination back to the source. Because there are often clock gaters along the clock paths and those clock gaters have latches along the enable paths, you have to let the enable values propagate through the latches to be able to disable the clock gaters and trace through the clock gaters.

Note

 When using this option, it is important that you later come and verify the proper connection and polarity of the latches along the enable paths to make sure they were implemented correctly.

Examples

The following example shows the effect of adding a force on a particular gate_pin object, and then issuing the simulate_forces command to trigger the tool to simulate that force:

```
ANALYSIS> set_current_simulation_context myContext  
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 1  
ANALYSIS> report_simulation_forces
```

Value	Object
-----	-----
1	I1

```
ANALYSIS> get_simulation_value_list [get_gate_pins I1]  
{ X }
```

```
ANALYSIS> simulate_forces  
ANALYSIS> get_simulation_value_list [get_gate_pins I1]  
{ 1 }
```

Related Topics

[add_simulation_context](#)
[get_simulation_context_list](#)
[add_simulation_forces](#)
[get_simulation_value_list](#)
[copy_simulation_context](#)
[report_simulation_contexts](#)
[delete_simulation_contexts](#)
[report_simulation_forces](#)
[delete_simulation_forces](#)
[simulate_clock_pulses](#)

[simulate_patterns](#)

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Performs simulation by applying the specified pattern source.

Usage

```
simulate_patterns [-source {external | bist | random}]  
[-store_patterns {all | effective | none}]
```

Description

Performs simulation by applying the specified pattern source.

The simulate_patterns command performs a good-machine simulation, and also performs a fault simulation only when there is a current fault list. By default, the command simulates the external patterns previously loaded with the [read_patterns](#) command, and the tool retains the pattern order (disables pattern classification) after simulation. The command optionally stores the patterns to the internal pattern set.

The command issues a warning before performing a good-machine simulation when no patterns are stored.

When executing the simulate_patterns command, the tool analyzes the design to determine if split capture cycle or clock_off simulation is needed. If so, the tool turns these functions on automatically. When the pattern source is set to BIST and an NCP (named capture procedure) is used, the tool checks the NCP to determine the necessary sequential depth and other environment settings (such as multi-load or hold_pi) before starting fault simulation. For more information, refer to “[Example 2](#)” on page 2476.

Arguments

- `-source {external | bist | random}`

An optional switch and literal pair that specifies which pattern source to simulate. By default, the command simulates the external patterns previously loaded with the [read_patterns](#) command.

The “bist” option specifies that the tool simulate only BIST patterns. Note that when running from BIST mode (LFSR is defined or in EDT_LBIST mode) and no external patterns are loaded, the default pattern source is BIST. There must be at least one defined LFSR. The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup.

You can use the `-store_patterns` switch to store the BIST patterns when performing good simulation; however, the “`-store_patterns effective`” option is not allowed. These patterns are different from normal patterns in that they specify the excess values that occur on short

scan chains during the load and unload process. Also, the last pattern does not contain an unload of the scan chains.

The “random” option specifies the random patterns, capture clock, and observation point specified previously. You can use this option to create patterns for random pattern testable faults. For fault simulation, you can use this option to evaluate the expected random pattern test coverage.

- **-store_patterns {all | effective | none}**

An optional switch and literal pair that specifies if the simulated patterns should be added to the internal pattern set. By default, the command stores no patterns. However, you can optionally store all patterns or store only the patterns that were effective at detecting additional faults.

Examples

Example 1

The following example performs a good-machine simulation using the current BIST patterns and then adds those patterns to the internal pattern set:

```
simulate_patterns -source bist -store_patterns all
```

Example 2

The following example applies an NCP named “cap_depth3” and then initiates a BIST pattern fault simulation. Note that before running the simulation, the tool first analyzes the NCP and then automatically changes the sequential depth to 3 and also issues a “set_split_capture_cycle on” command.

```
add_bist_capture_range cap_depth3 0 511
simulate_patterns -source bist
```

```
// Warning: Current sequential depth is not large enough to apply named capture procedures.  
//           Automatically increase the sequential depth to be 3.  
// largest sequential test depth = 1, largest controllability depth = 1,  
// largest observability depth = 1  
// largest test depth is on /tck (57) (control = 0, observe = 1)  
// | -----  
// |       Current split capture setting:      Off  
// |           DRC requirement:      Yes (C3)  
// |           Calling:      set_split_capture_cycle on  
// |  
// |       Current clock off simulation setting:      Off (optimal)  
// | -----  
// |  
// -----  
// Simulation performed for #gates = 9002 #faults = 25443  
// Run fault simulation without storing patterns. pattern source = 512 BIST patterns  
// -----  
// #patterns  test      #faults  #faults    # eff.    # test      process  
// simulated coverage  in list   detected   patterns  patterns  CPU time  
...  
// 64        47.85%    12158     13285      64          0          0.50 sec  
// 128       53.66%    10121     2037       56          0          0.56 sec  
// 192       56.52%    9118      1003       38          0          0.60 sec  
...
```

Related Topics

[delete_patterns](#)

[write_patterns](#)

[read_patterns](#)

sizeof_collection

Context: unspecified, all contexts

Mode: all modes

Returns the number of objects in a collection.

Usage

`sizeof_collection collection`

Description

Returns the number of objects in a collection.

The `sizeof_collection` command is an efficient way to determine how many objects are in a collection.

Arguments

- *collection*

A required string that specifies the collection for which to get the number of objects. The returned size for an empty collection is 0.

Examples

Example 1

The following example shows a simple way to find out how many instances of cell module the current design has.

```
puts [sizeof_collection [get_instances -filter {type == "cell"}]]
```

```
56781
```

Example 2

The following example shows the size of a collection as it gets appended to.

```
set a ""
puts [sizeof_collection $a]
0

append_to_collection a b[7]
{b [7] }

puts [sizeof_collection $a]
1

append_to_collection a {b c d}
{b [7] b c d}
```

puts [sizeof_collection \$a]

4

Related Topics

[add_to_collection](#)
[append_to_collection](#)
[compare_collections](#)
[copy_collection](#)
[filter_collection](#)
[foreach_in_collection](#)
[index_collection](#)
[is_collection](#)
[range_collection](#)
[remove_from_collection](#)
[sort_collection](#)

sort_collection

Context: unspecified, all contexts

Mode: all modes

Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

Usage

`sort_collection collection criteria [-descending] [-numerical_sort]`

Description

Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

The sort is in ascending order by default. In an ascending sort, objects that have Boolean attributes set to false are first followed by Boolean objects that have attributes set to true. The sorting considers the default values of the attributes when the attribute has never been set on the object.

To sort the objects in reverse order, specify the `-descending` option. To sort the objects numerically, instead of lexically, specify the `-numerical_sort` option.

Arguments

- ***collection***

A required string that specifies the collection to be sorted.

- ***criteria***

A required value that specifies an attribute name or a Tcl list of attribute names to use as sort keys. The order of the attribute name in the list controls the order of priority while sorting.

- `-descending`

An optional switch that sorts the collection in reverse order. By default, the sort proceeds in ascending order.

- `-numerical_sort`

An optional switch that sorts the collection numerically.

This is the sorted collection with the switch:

```
core1_28x16/a[1]
core1_28x16/a[2]
core1_28x16/a[3]
core1_28x16/a[10]
core1_28x16/a[20]
```

By default, the sort proceeds lexically. This is the sorted collection without the switch:

```
core1_28x16/a[10]
core1_28x16/a[1]
core1_28x16/a[20]
core1_28x16/a[2]
core1_28x16/a[3]
```

Examples

The following example illustrates the use of two attributes to sort a collection. Notice how the first attribute has higher priority than the second attribute in the sorting.

```
register_attribute -name att2 -value_type string -default a
register_attribute -name att1 -value_type string -default a

set_attribute_value {ba bb} -name att1 -value b
{ba bb}

set_attribute_value {ab bb} -name att2 -value b
{ab bb}

sort_collection [get_inst {aa ab ba bb}] {att1 att2}
{aa ab ba bb}

sort_collection [get_inst {aa ab ba bb}] {att2 att1}
{aa ba ab bb}

sort_collection [get_inst {aa ab ba bb}] {att2 att1} -descending
{bb ab ba aa}
```

Related Topics

[add_to_collection](#)
[compare_collections](#)
[filter_collection](#)
[index_collection](#)
[is_collection](#)
[sizeof_collection](#)
[append_to_collection](#)
[copy_collection](#)
[foreach_in_collection](#)
[range_collection](#)
[remove_from_collection](#)

start_silicon_insight

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Starts Tesson SiliconInsight and initializes the data structures.

Usage

`start_silicon_insight [-gui]`

Arguments

- `-gui`

Starts the Tesson SiliconInsight GUI.

stop_silicon_insight

Context: patterns -silicon_insight

Mode: setup

Stops Tesson SiliconInsight.

Usage

`stop_silicon_insight`

Description

Closes the CDP and exits the sid_tester process when you are running Tesson SiliconInsight Desktop, SimDUT or “NoTester.” When you are running on an ATE, this command closes the connection to the test program.

Arguments

None

system

Context: unspecified, all contexts

Mode: all modes

Passes the specified command to the operating system for execution.

Usage

`system os_command`

Description

Passes the specified command to the operating system for execution.

The system command executes one operating system command without exiting the currently running application.

Note

 There have been isolated incidents in which the “system echo” command resulted in memory overrun. If this occurs, the recommendation is to use the TCL “puts” command instead.

Arguments

- *os_command*

A required string that specifies any legal operating system command.

Examples

Example 1

The following example invokes a synthesis script to synthesize generated RTL, and reloads the synthesized view:

```
write_design -output_file rtl/my_module.v
system "cd synth_work; ./synth_script"
read_verilog gate/my_modul.v -force
set_current_design mytop
```

Example 2

The following example performs an ATPG run, then displays the current working directory without exiting the tool:

```
set_system_mode analysis
create_patterns
system pwd
```

Related Topics

[setenv](#)

unsetenv

trace_flat_model

Context: all contexts

Mode: setup, analysis

Prerequisites: This command can only be used after the design is flattened to the simulation model; this occurs when you first exit setup mode to enter analysis mode, or when you issue the [create_flat_model](#) command.

Traces the flat model taking into account the current simulation context.

Usage

```
trace_flat_model -from from_obj_spec [-to to_obj_spec] -direction {forward | backward}
[-tag_condition tag_attribute_expression] [-stop_condition stop_attribute_expression]
[-controllability {controlling | controlling_allow_reconvergence | constant | unblocked
| connected}] [-stop_at_first_tag {On | Off}] [-max_levels {unlimited | number}]
[-max_combinational_cells {unlimited | number}] [-max_tags {unlimited | number}]
[-latch {normal | transparent}] [-map_tag_to_design_module_boundary {ON | OFF}]
[-store_polarity_to_tagged_objects polarity_var_name]
[-store_polarity_along_paths_to_tagged_objects path_polarity_var_name]
[-store_paths_to_tagged_objects tag_path_var_name]
[-store_path_to_stop_point stop_path_var_name]
[-store_starts_to_tagged_objects starts_to_tagged_objects_var_name]
```

Description

Traces the flat model taking into account the current simulation context.

For information about simulation contexts, refer to the [set_current_simulation_context](#) command description.

The tool begins tracing at the objects specified by the -from option. The direction is either forward or backward. You can use equations based on attributes found on gate_pin objects to specify the gate_pins that you want the tracing to stop on, or the gate_pin objects you want to tag when reached.

This command returns a collection of reached gate_pins for which the tag condition evaluates to true. When the -tag_condition argument is specified, this command returns a collection of all gate_pin objects the trace stopped on. You can also provide a Tcl variable in which to store the path between the start object and the tagged objects so that you can display them in DFTVisualizer as shown in two of the following examples.

Note

 This command only returns the start point when it is reached again during the trace, that is, when there is a feedback path.

When using this command in setup mode, you may see the following error message because no simulation data is available:

```
Error: Controllability 'unblocked' is the default, but no simulation  
data exists in the current simulation context ...
```

To avoid this error, you must specify the “-controllability connected” trace option.

Arguments

- **-from *from_obj_spec***

A required option that specifies a Tcl list of one or more names of gate_pin, pin, or port objects, or a collection of one or more gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

Pins that are preserved in the flat model are either all pins on instances of type cell, or pins on instances of type design on which an attribute is set to a non-default value and the set_attribute_option -preserve_boundary_in_flat_model option of that attribute is set to true. These attributes must have been defined on the pin in setup mode prior to creating the flat model.

Note

 By default, when the tool flattens the design, hierarchical pins are not kept as gate_pin elements. If you need to work with a hierarchical pin, you need to instruct the tool to keep it as part of the flat model by using the “preserve_boundary” option as shown here: set_attribute_value *object_spec* -name preserve_boundary

The tracing is performed starting from each object specified in the *from_obj_spec* argument.

When the direction is forward and the start object has no fanout, a null collection is returned. When the direction is backward and the start object has no fanin or is tied to a constant, a null collection is returned when the -map_tag_to_design_module_boundary switch is set to on as the tieX or tie0 or tie1 gate in the fanin does not correspond to a design module boundary.

- **-to *to_obj_spec***

An optional switch and value pair that directs the trace operation to stop at the objects specified by *to_obj_spec* and to tag them, if reached. The *to_obj_spec* argument can be a Tcl list of one or more names of gate_pin, pin, or port objects, or a collection of one or more gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

You can use this switch to specify objects as stop points in addition to the objects specified with the -stop_condition switch, and to specify which objects will be tagged in addition to the objects specified with the -tag_condition switch.

- **-direction {forward | backward}**

A required string that specifies whether tracing should be performed forward or backward from the defined start object(s).

- **-tag_condition tag_attribute_expression**

An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the *-filter* option used for the [get_pins](#) command.

- **-stop_condition stop_attribute_expression**

An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the *gate_pin* failing the specified controllability requirements. The pins of memory or dff primitives are always stop points even when not explicitly specified.

- **-controllability {controlling | controlling_allow_reconvergence | constant | unblocked | connected}**

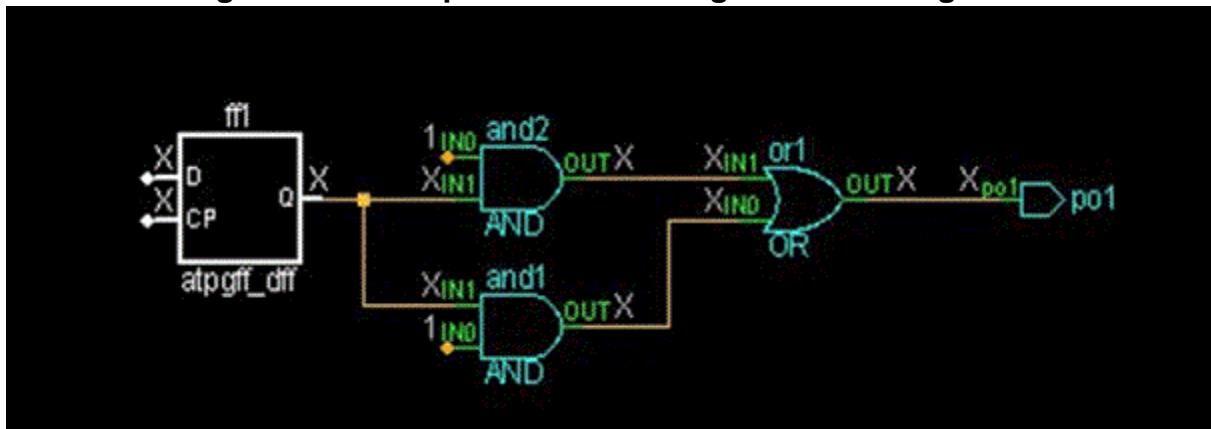
An optional switch and literal pair that defines the controllability requirements before traversing a gate.

controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output. Inversions are possible and information about them is stored in the variable specified by the *-store_polarity_to_tagged_objects* argument.

controlling_allow_reconvergence — Supports tracing reconvergent controlling paths. When this option is specified, the tool first tries to perform the trace with controllability 'controlling' (see description above). If this trace succeeds, the tool behavior is identical with a trace performed with the controllability controlling. If this trace fails, an unblocked backward trace from the target is internally performed to find all the possible controlling source candidates. These candidates are then investigated by simulating 0 and 1 to them and observing the effect on the destination.

In [Figure 6-6](#) on page 2488, you can see one example of a reconvergent controlling path from ff1/Q to po1.

Figure 6-6. Example of a Reconvergent Controlling Path



Information about inversions is stored in the variable specified by the -store_polarity_to_tagged_objects switch. In case the reconvergence analysis was necessary to find the controlling source, the variable specified by the -store_paths_to_tagged_objects switch returns an empty collection.

Unnecessarily specifying controllability controlling_allow_reconvergence instead of controlling has negative impact on tool's performance.

constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected — With this value, gates are always traversed when reached. You should use this option when this command is used in setup mode because no simulation data exists yet.

- **-stop_at_first_tag {On | Off}**

An optional switch and literal pair that specifies whether to interrupt tracing when the first tag object with the specified controllability is found. This is useful when checking for at least one occurrence of a given condition in the fanin or fanout of a pin without wasting time to find them all.

- **-max_levels {unlimited | number}**

An optional switch and value pair that defines the maximum levels of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count. The gates are the primitive gates as seen in the flat model. An AND-OR gate will have two levels of primitive gates in the flat model and will count as two levels of gates for this limit. If you want to only count cells irrespective of their complexity, use the -max_combinational_cells switch instead.

- **-max_combinational_cells {unlimited | number}**

An optional switch and value pair that defines the maximum number of combinational cells to trace. The default is unlimited. Each combinational cell is counted as one cell except for inverters and buffers which do not count at all. Even if a cell has several levels of primitives inside it, it is not counted as more than one cell. To limit the search to a maximum number of primitive gates, use the -max_levels switch instead.

- **-max_tags {unlimited | number}**

An optional switch and value pair that directs the trace operation to stop when the number of tagged objects reaches the specified limit. The default is unlimited. This is useful when you are coding your own custom DRC using the [register_drc](#) register_drc command and you only want detect the first 100 violations. There is no point tagging all violations if you only want to show the first 100.

- **-latch {normal | transparent}**

An optional switch and literal pair that specifies the behavior of tracing through a latch.

normal — A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of “-controllability connected” all latch input pins - including the latch enable pin - are traced through.

transparent — All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- **-map_tag_to_design_module_boundary {ON | OFF}**

An optional switch and literal pair that specifies whether to map the tag points to `gate_pins` outside of library cells. When this switch is ON, the command maps the tag points to `gate_pins` outside of library cells and never inside the cells. Library cells can be either cells from a Tesson Cell library or `celldefine Verilog modules.

- **-store_polarity_to_tagged_objects *polarity_var_name***

An optional switch and string pair that specifies the name of a Tcl variable in which to store a list of 0 or 1 values to reflect the presence of an inversion between the start object and the tag object(s). If an inversion exists between the start and tag object(s), the value 1 is stored in the list; otherwise, the value 0 is stored. This option can only be used with the -controllability controlling option; inversion is meaningless in all other cases.

- **-store_polarity_along_paths_to_tagged_objects *path_polarity_var_name***

An optional switch and string pair that specifies the name of a Tcl variable in which to store a list of 0 or 1 values to reflect the presence of inversions between the start object and the tag object(s). If an inversion exists between the start and tag object(s), the value 1 is stored in the list; otherwise, the value 0 is stored. This switch provides a list of 0 and 1 values representing the local polarity of each node that the corresponding collection found by the -store_paths_to_tagged_objects switch. This option can only be used with the -controllability controlling option; inversion is meaningless in all other cases.

- **-store_paths_to_tagged_objects *tag_path_var_name***

An optional switch and string pair that specifies to store a list of paths in the variable `tag_path_var_name`. Each path holds a collection containing all of the traversed `gate_pins` between the start object and the tagged object(s). The list is in the order specified by the -direction option and includes the start object and the reached tagged object.

- **-store_path_to_stop_point *stop_path_var_name***

An optional switch and string pair that specifies to store the path between the start object and the `gate_pin` where the trace stopped independent of whether the stop point is a tagged object. This switch is only valid when -direction is specified as backward and -controllability is specified as controlling or constant. In this case, only one start object is supported and only one stop point can ever be found. This mode is useful to trace a clock source and display the path to the stop point if an appropriate source is not reached.

- **-store_starts_to_tagged_objects *starts_to_tagged_objects_var_name***

An optional switch and string pair that specifies to store the start point from which all tagged objects were reached from. The return value is a single collection containing the start points from which the tagged objects were seen from.

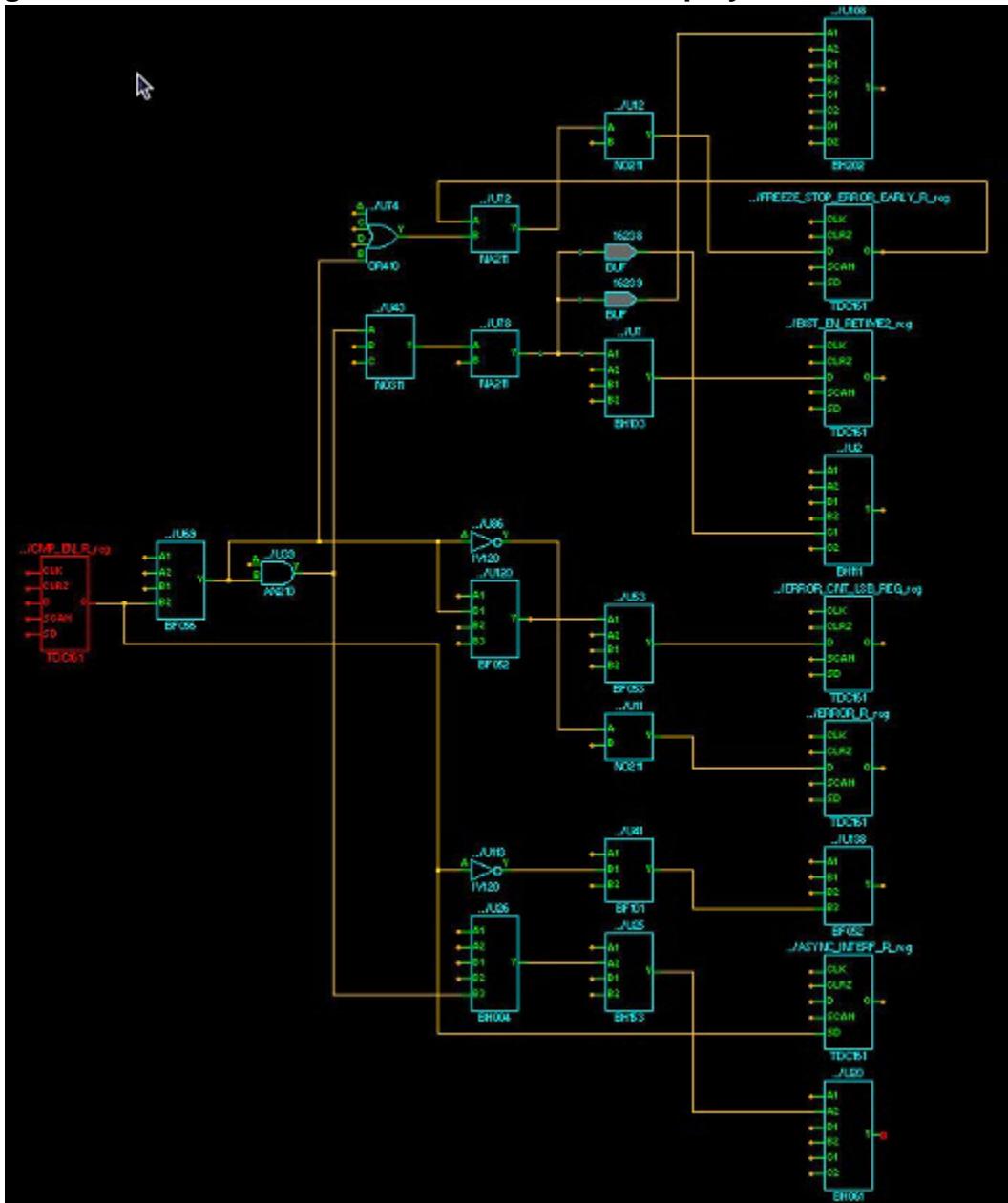
Examples

Example 1

The following example traces the unblocked fanout of a pin. Because no `-tag_condition` is specified, the tool automatically tags every node with no unblocked fanout. The paths to the tag elements are stored in the Tcl variable `tag_paths`. The `foreach` command iterates on the list of paths and the `add_display_instance` command displays the paths in the DFTVisualizer Flat Schematic window. [Figure 6-7](#) shows the resulting schematic drawn in the Flat Schematic window.

```
set start_pin [get_pins BOB_CKB2_1_MBIST1_MBIST_I1/CMP_EN_R_reg/Q]
trace_flat_model -from $start_pin -direction forward \
    -store_paths_to_tagged_objects tag_paths
add_display_instance $tag_paths
mark_display_instances $start_pin -marking_index 5
```

Figure 6-7. Flat Schematic Window After add_display_instance Command

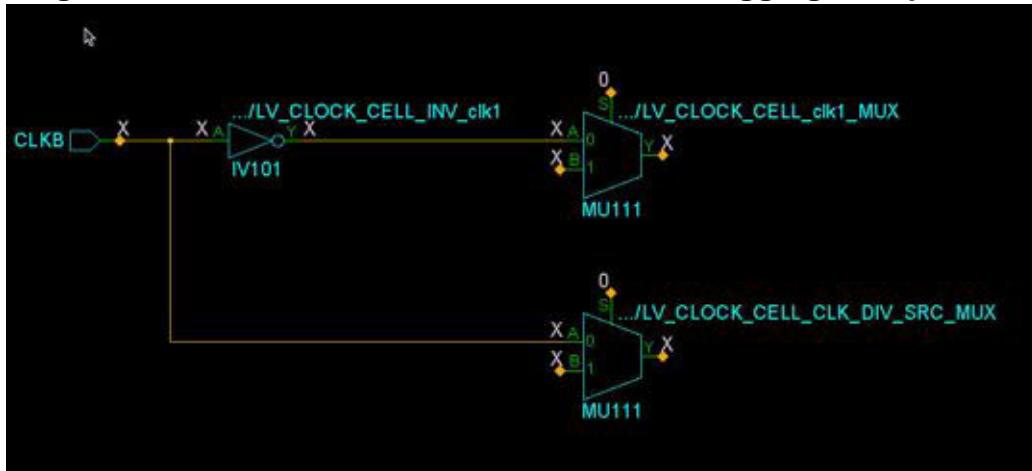


Example 2

The following example performs a controlling walk forward from a clock source and collects any MU111 multiplexers traversed along the way. The paths from the start point to all tag elements are displayed in the Flat Schematic window as shown in [Figure 6-8](#). Notice how the set_gate_report drc command is set to stable_after_setup; this corresponds to the simulation conditions used by the trace_flat_model command. You can see that the two muxes have a controlling value on the select pin. Multiplexers without a controlling value on the select pin were not traversed and therefore their output pins were not tagged.

```
trace_flat_model -from CLKB \
    -tag_condition {module_name == MU111 && direction == output} \
    -direction forward \
    -store_paths_to_tagged_objects tag_paths -controllability controlling
set_gate_report drc stable_after_setup
add_display_instance $tag_paths
```

Figure 6-8. Result of trace_flat_model While Tagging Multiplexers



Example 3

The following example uses the trace_flat_model command to find paths between output and inout pins of modb to pins of modd. Notice how the -stop_at_first_tag option is set to on, ensuring that only one path from each modb pin is reported.

```
set_system_mode setup
set_attribute_value [get_instances -of_module {modb modd}] -name preserve_boundary
set_system_mode analysis
trace_flat_model -from [get_pins u4/* -direction {output inout}] -direction forward \
    -stop_condition preserve_boundary \
    -tag_condition {module_name == modd} \
    -stop_at_first_tag on \
    -store_paths_to_tagged_objects paths
if {[llength $paths] > 0} {
    puts "pins of modb fanning out to modd:"
    foreach path $paths {
        puts [get_single_name [index_collection $path 0]]
    }
}
```

Related Topics

[get_trace_flat_model_option](#)
[set_current_simulation_context](#)
[set_trace_flat_model_options](#)

uncompress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Uncompresses a Layout Database.

Usage

LDBs created after v2013.2: `uncompress_layout`

LDBs created with v2013.2 and before: `uncompress_layout layout_database_name`

Description

Uncompresses a Layout Database.

You must open the Layout Database using the [open_layout](#) command before performing the uncompress operation.

See “[Layout Database Compression and Decompression](#)” in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- *layout_database_name*

A required string that specifies the location to and the name of the Layout Database to be created during the uncompress operation.

Related Topics

[compress_layout](#)

[create_layout](#)

[open_layout](#)

uniquify_instances

Context: all contexts

Mode: insertion

Uniquifies the module definition of the specified instance objects found in *obj_spec*.

Usage

```
uniquify_instances obj_spec [-module_uniquification_suffix module_uniquification_suffix]  
[-silent]
```

Description

Uniquifies the module definition of the specified instance objects found in *obj_spec*.

If the instances specified by *obj_spec* have a module definition that is used more than once below the current design, the module name is made unique by appending the specified -module_uniquification_suffix value to its name.

This command returns a collection of all modules associated with the instances in the *obj_spec* irrespective of whether the module name was uniquified or not.

Arguments

- *obj_spec*
A required value that specifies the instances whose module name is to be uniquified. The *obj_spec* value must be a Tcl list of one or more instance names, or a collection of one or more instances as generated by the get_instances command
- -module_uniquification_suffix *module_uniquification_suffix*
An optional switch and value pair that specify a string to append to the module name if the instance's module definition is used more than once below the current design. The default module_uniquification_suffix is “_#” where # begins at 1 and increments by one until a unique module name is found.
- -silent
An optional argument that specifies to ignore specified instance objects that do not exist or instances that are not of type design, and to not generate error messages.

Examples

Example 1

The following example requests all instances found within u2 to be uniquified. All instances of design modules in this example are instances of module moda and modb. From the result of the command, you can see that there are only three instances of moda within the current design and they are all within the u2 instance. Once the first two instances were uniquified to moda_1 and moda_2, the third instance was not changed as it was now unique. Because modb was uniquified, this means that it is used in other instances of the current design.

```
uniquify_instances [get_instances u2/* -of_type design ]
{moda moda_1 moda_2 modb_1}
```

Example 2

The following example overwrites the default module_uniquification_suffix of “_#” to “_uniq#” so that it is easier to identify which modules were uniquified:

```
uniquify_instances [get_instances u2/* -of_type design] -module_unification_suffix _uniq#
{moda moda_uniq1 moda_uniq2 modb_uniq1}
```

Example 3

The following example appends to the my_design_modules collection the collection of modules returned by the uniquify operation. Note that by definition, uniquify_instances will return a collection with a different module object for each instances object because uniquified modules are only ever instantiated once.

```
append_to_collection my_design_modules \
[uniquify_instances [get_instances u2/* -of_type design]]
```

Related Topics

[create_instance](#)
[delete_instances](#)
[get_common_parent_instance](#)
[rename_instance](#)
[replace_instances](#)

unmark_display_instances

DFTVisualizer windows: Debug, Design

Context: all contexts

Mode: setup, analysis

Returns the specified marked instances to their original color. Color marking occurs with the [mark_display_instances](#) command.

Usage

```
unmark_display_instances {gate_id# | instance_name | pin.pathname}... | -All | -Selected  
[-Display {FLAt_schematic | HIHierarchical_schematic}]
```

Description

Returns the specified marked instances to their original color. Color marking occurs with the [mark_display_instances](#) command.

Tip

 If a gate is selected, any highlighting or marking applied to the gate is not visible until the gate is unselected.

Arguments

- ***gate_id#***

A repeatable integer that specifies the gate identification number of instances to unmark display instances. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- ***instance_name***

A repeatable string that specifies the name of an instance to unmark display instances.

- ***pin.pathname***

A repeatable string that specifies the name of a pin whose gate you want to unmark display instances.

- **-All**

A switch that specifies to unmark all the gates in the design.

- **-Select**

A switch that specifies to unmark currently selected gates.

- -Display {FLAt_schematic| HIEarchical_schematic}

A switch and literal pair that specifies the window for which to apply the unmark_display_instances command. The following literals can be used with the -Display switch:

FLAt_schematic — Applies the settings to the Flat Schematic window. This is the default behavior.

HIEarchical_schematic — Applies the settings to the Hierarchical Schematic window.

Examples

The following example unmarks two instances by specifying the instance name of one and the gate identification number (85) of the other:

unmark_display_instances /i\$142 85

Related Topics

[mark_display_instances](#)

[open_visualizer](#)

[select_display_instances](#)

unregister_attribute

Context: unspecified, all contexts

Mode: all modes

Unregisters a user-defined attribute that was registered using the register_attribute command.

Usage

```
unregister_attribute -name attribute_name [-object_types type_list] [-silent]
```

Description

Unregisters a user-defined attribute that was registered using the register_attribute command.

Following de-registration, the attribute is unavailable for use.

Pre-defined attributes cannot be de-registered.

If attribute_name does not exist, or if a design object type in type_list does not exist, an error message displays unless the -silent is specified.

Arguments

- **-name *attribute_name***

A required switch and value pair that specifies the name of the attribute to de-register.

- **-object_types *type_list***

An optional switch and value pair that specifies the object type to which the new attribute can be assigned. The type_list string is a Tcl list that contains one or more of the following object types: module, instance, port, pin, or net. If an object type is not included in the type_list, the new attribute cannot be assigned to that object_type.

If this argument is not specified, by default, the new attribute is de-registered for all object types.

- **-silent**

An optional switch that suppresses error messages if the given attribute_name is not registered.

Examples

The following example de-registers the attribute added_by for only the net object type.

```
unregister_attribute -name added_by -object_type net
```

Related Topics

[get_attribute_list](#)

[get_attribute_option](#)

[get_attribute_value_list](#)

[register_attribute](#)

[report_attributes](#)

[reset_attribute_value](#)

[set_attribute_options](#)

[set_attribute_value](#)

[unregister_static_dft_signal_names](#)

Context: unspecified, all context

Mode: all modes

Prerequisites: You cannot unregister static dft signal names after the [add_dft_signals](#) command is run.

Registers previously registered static DFT signal names.

Usage

```
unregister_static_dft_signal_names {name ...} | -all_user_defined
```

Description

Registers previously registered static DFT signal names. After you have run the [add_dft_signals](#) command, it is no longer possible to unregister a static DFT signal. Only DFT signal previously registered with the [register_static_dft_signal_names](#) command can be unregistered. The built-in names as well as the logic test control signals registered using the [tcd_scan](#) metadata cannot be unregistered.

Arguments

- ***name***

A repeatable string value that specifies the names of the DFT signals to unregister. The string can also be a well-formatted Tcl list of names. These names must correspond to DFT signal names that were registered using the [register_static_dft_signal_names](#) command.

- **-all_user_defined**

A Boolean switch specifying that all DFT signal names previously registered using the [register_static_dft_signal_names](#) command are to be unregistered.

Examples

The following example registers a DFT signal name then unregisters it:

```
register_static_dft_signal_names clockb_select  
unregister_static_dft_signal_names clockb_select
```

Related Topics

[register_static_dft_signal_names](#)

[report_dft_signal_names](#)

unregister_tcl_command

Context: unspecified, all contexts

Mode: all modes

Unregisters a command that was previously registered using the register_tcl_command.

Usage

`unregister_tcl_command command_name`

Description

Unregisters the command that was previously registered using the register_tcl_command.

An error is generated if the command was registered before the last lock_current_registration command invocation. Commands loaded as part of a plugin are automatically locked.

Arguments

- *command_name*

A required string that specifies the command to unregister.

Examples

The following example unregisters the commands ABC and DEF. The command ABC was registered prior to the last lock_current_registration command so it cannot be unregistered:

```
unregister_tcl_command ABC
// Error: Command locked and can't be un-registered

unregister_tcl_command DEF
```

Related Topics

[lock_current_registration](#)

[register_tcl_command](#)

unselect_display_instances

Context: all contexts

Mode: setup, analysis

Unselects the specified objects in the Flat and Hierarchical Schematic window(s).

Usage

```
unselect_display_instances {{gate_id# | instance_name | pin.pathname}} ... | -All  
[-Display {FLAt_schematic | HIErarchical_schematic}]
```

Description

Unselects the specified objects in the Flat and Hierarchical Schematic window(s).

Arguments

- ***gate_id#***

A repeatable integer that specifies the gate identification number of a gate to unselect. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- ***instance_name***

A repeatable string that specifies the name of an instance to unselect.

- ***pin.pathname***

A repeatable string that specifies the name of a pin associated with a gate you want to unselect.

- **-ALL**

A switch that unselects all currently displayed gates.

- -Display {FLAt_schematic | HIErarchical_schematic}

A switch and literal pair that specifies the window in which this command selects objects. The following literals can be used with the -Display switch:

FLAt_schematic — Unselects the specified objects in the Flat Schematic window. This is the default behavior.

HIErarchical_schematic — Unselects the specified objects in the Hierarchical Schematic window.

Examples

The following example unselects an object identified by its instance name in the Flat Schematic window:

```
unselect_display_instances /i_top/i144
```

Related Topics

[open_visualizer](#)

[select_display_instances](#)

unsetenv

Context: unspecified, all contexts

Mode: all modes

Unsets a shell environment variable within the tool environment.

Usage

`unsetenv variable_name`

Description

Unsets a shell environment variable within the tool environment.

The unsetenv command unsets a shell environment variable within the current tool environment. When you only need a variable unset within the tool, this command saves you the time and trouble of exiting, issuing the UNIX unsetenv command in the shell, and re-invoking.

Note

 A shell variable you unset with this command is unset only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To set or change the value of a variable, use the [setenv](#) command. To view the value of the variable, use the [system](#) command to pass an appropriate shell command ([echo](#) or [env](#) for example) to the operating system for execution.

Arguments

- *variable_name*

A required string that specifies the name of the variable.

Examples

The following example unsets the SSH_AGENT_PID environment variable within the current tool environment, then confirms that it is unset:

```
system {echo $SSH_AGENT_PID}  
66216  
  
unsetenv SSH_AGENT_PID  
system {echo $SSH_AGENT_PID}
```

Related Topics

[setenv](#)

[system](#)

update_implicit_detections

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

Usage

```
update_implicit_detections
```

Description

Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

The update_implicit_detections command is automatically issued after the simulate_patterns command executes a fault simulation when using external patterns or when using internal patterns in non-Atpg. The update_implicit_detections command is also automatically issued as part of the create_patterns command.

By invocation default, only scan-path-associated faults for the detected-by-implication classification are analyzed. The following faults are classified as detected-by-implication when the update_implicit_detections command is issued:

- A stuck-at-1 fault on the set input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the output.
- A stuck-at-1 fault on the reset input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-0 fault on the output.
- A stuck-at-0 fault on a clock input line of a transparent latch, scan latch, scan flip flop, shadow, copy, or sequential cell when the tool detects both the stuck-at-0 and stuck-at-1 faults for the associated data line.
- A stuck-at-0 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy or sequential cell when the tool detects the stuck-at-0 fault and no other ports can capture a 1.
- A stuck-at-1 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the data output and no other port can capture a 0.

Arguments

None

Related Topics

[report_faults](#)

verify_patterns

Context: patterns -scan_diagnosis

Mode: analysis

Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

Usage

```
verify_patterns {-Diagnosis |
  -Create_startup_cache startup_cache_name [-Force_open] |
  -Update_startup_cache startup_cache_name [-Force_open] |
  -Load_startup_cache startup_cache_name}
```

Description

Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

See “[Pattern Verification and the Diagnosis Startup Cache](#)” in the *Tessent Diagnosis User’s Manual* for complete information.

Arguments

- **-Diagnosis**

A required switch that only runs pattern verification without storing the pattern verification results into Diagnosis Startup Cache. The diagnosis tool outputs the pattern verification results to stdout and in the tool’s transcript.

If the pattern verification passes, then Tessent Diagnosis does not re-run pattern verification following the [diagnose_failures](#) command, unless there changes to the current pattern set.

- **-Create_startup_cache *startup_cache_name***

A required switch and string pair that runs pattern verification and creates a new Diagnosis Startup Cache. If *startup_cache_name* already exists, then the tool produces an error.

If the pattern verification passes, then the tool creates a new Diagnosis Startup Cache with the given filename. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not create the Diagnosis Startup Cache.

- **-Update_startup_cache *startup_cache_name***

A required switch and string pair that loads an existing Diagnosis Startup Cache and compares the contents of the Diagnosis Startup Cache (netlist, patterns, and so on) to this information in the current Tessent Diagnosis session. If *startup_cache_name* does not exist, then the tool produces an error.

If the specified Diagnosis Startup Cache does not contain the current pattern sets or does not match the current tool version, then the tool runs pattern verification.

If the pattern verification passes, then the tool updates the specified Diagnosis Startup Cache. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not update the Diagnosis Startup Cache.

- **-Load_startup_cache *startup_cache_name***

A required switch and string pair that loads an existing Diagnosis Startup Cache. If *startup_cache_name* does not exist, then the tool produces an error and exits.

If the specified Diagnosis Startup Cache contains the current pattern sets and matches the current tool version, then the tool loads the Diagnosis Startup Cache. Otherwise, the tool produces an error.

- **-Force_open**

An optional switch used with either -Create_startup_cache or -Update_startup_cache that unlocks a locked Diagnosis Startup Cache.

Examples

Example 1

This example demonstrates creating a Diagnosis Startup Cache for future use in diagnosis with Tessian Diagnosis.

In the Tessian Diagnosis scan diagnosis point tool, the following command sequence loads the test patterns, performs the pattern verification, and creates a Diagnosis Startup Cache named *my_startup_cache*:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -create_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessian Diagnosis, load the patterns and newly created Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
.....
exit
```

Example 2

This example command sequence illustrates updating an existing Diagnosis Startup Cache (*my_startup_cache*) for future use by Tessian Diagnosis:

```
read_patterns pat1
read_patterns pat2 -append
read_patterns pat3 -append
verify_patterns -update_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessian Diagnosis, load the patterns and updated Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 -append
read_patterns pat3 -append
verify_patterns -load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
.....
exit
```

Example 3

The following example provides the command sequence you would perform to use an existing Diagnosis Startup Cache with the Tessian Diagnosis Server:

1. Create the Diagnosis Startup Cache (*my_startup_cache*) with the following commands using the Tessian Diagnosis scan diagnosis point tool:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -create_startup_cache ./my_startup_cache
exit
```

2. In the Tessian Diagnosis Server, add the patterns and load the Diagnosis Startup Cache (*my_startup_cache*) using the following command sequence:

```
add_pattern monitor1 pat1
add_pattern monitor1 pat2
add_startup_cache monitor1 ./my_startup_cache
...
```

Example 4

In this example, Tesson Diagnosis does not create a Diagnosis Startup Cache or re-run pattern verification during diagnosis:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns --diagnosis // assume pattern verification passed
diagnose_failures log1 // will not re-run pattern verification
```

Related Topics

[diagnose_failures](#)
[read_patterns](#)

write_atpg_setup

Context: dft -scan

Mode: analysis, insertion

Prerequisites: Scan chains must be inserted.

Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.

Usage

```
write_atpg_setup basename [-Replace] [-NO_merge] [-NO_Edt] [-All_internal_clocks]
```

Description

Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.

You can use these files to identify the scan chains when loading the scan design into Tessent FastScan. By default, the test procedure and the dofile for newly inserted scan chains are merged with those for existing scan chains.

The order of the commands in the test procedure file determine the different levels of test procedure file information produced by this command.

- If you execute write_atpg_setup without first using the [read_procfle](#) command, a test procedure file that contains default timeplates and the procedures needed for ATPG setup (for example, shift, load_unload, and test_setup) are created.
- If you run [read_procfle](#) and load a test procedure file that contains timeplate definitions, non-scan procedure definitions (capture, ram_sequential, and others), and empty or template scan procedures, and then execute the write_atpg_setup command, a more complete test procedure file is created.

Note

 If the test procedure file loaded with the [read_procfle](#) command contains scan procedures (for example, shift and load_unload), they are replaced by ones generated with the write_atpg_setup command based on test logic inserted during the current session. The tool issues a W14 warning each time it replaces a procedure. However, it uses timeplates specified in the original procedure file for the newly-generated procedures. The only exceptions are for the test_setup procedure. Instead of being replaced, the events created by the write_atpg_setup command are appended to the end of the test_setup procedure.

By default, the tool automatically detects a SoC DFT flow and writes out EDT-specific commands to the ATPG setup files.

The dofile defines a Tcl procedure named “tessent_scan_setup” that accepts a parameter that is used to specify the name of the scan mode that you want to set up. For example, to define the scan chains that are accessible in “unwrapped” mode, you must add a call to tessent_scan_setup with the parameter set to “unwrapped”, as shown below:

```
dofile setup.dofile
tessent_scan_setup unwrapped
```

Finally, for designs that had pre-inserted scan chains, the default is that these scan chains will not be defined regardless of the selected mode. However, there is an optional argument to the tessent_scan_setup procedure that can be set to “1” if you want to also define these chains for the subsequent step. The example for this is shown below:

```
dofile setup.dofile
tessent_scan_setup unwrapped 1
```

In addition to generating the setup dofile, the tool will also generate a verification dofile that will enter the patterns -scan context, and will set up each scan mode, and trigger a system mode transition to verify that the inserted scan chains can be traced in that mode. By default, this procedure does not include any pre-defined chains. For designs with pre-existing chains, two additional procedures are created: verify_all_scan_modes_with_existing_chains and verify_all_scan_modes_without_existing_chains. You can choose the desired procedure after sourcing the verification dofile.

Hybrid TK/LBIST Flow Specifics

The regular scan insertion dofile includes additional commands that specify the pin names for scan enable and BIST enable signals. The tool automatically includes logicBIST-related commands either X-bounding or bounding of timing exceptions paths is implemented. Refer to the [Hybrid TK/LBIST Flow User’s Manual](#) for complete information.

Arguments

- ***basename***
A required string that specifies the root name for the test procedure and dofile. The files produced are *basename*.testproc and *basename*.dofile.
- **-Replace**
An optional switch that replaces the contents of a file that already exists. By default, files are not replaced.
- **-No_merge**
An optional switch that prevents merging the test procedure and the dofile of the newly inserted scan chains with those of the existing scan chains. By default, the test procedures and dofiles are merged.
- **-NO-Edt**
An optional switch that specifies that EDT-specific commands are *not* written to the ATPG setup files. By default, the tool automatically detects a SoC DFT flow and writes out EDT-

specific commands to the ATPG setup files. For more information, see the [set_edt_options](#) command.

- **-All_internal_clocks**

An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Examples

The following example writes the test procedure, the dofile, and the new netlist for the inserted scan chains to the specified filenames:

```
add_clocks 1 clk1
add_clocks 0 clk0
set_system_mode analysis
insert_test_logic

write_atpg_setup scan -replace
write_design -output_file scan.v
```

Related Topics

[insert_test_logic](#)
[read_procfile](#)
[set_edt_options](#)
[write_design](#)
[write_procfile](#)

write_cell_library

Tools Supported: Tessent Scan, Tessent FastScan, and Tessent TestKompress

Mode: all modes

Writes out a single library file of all library files loaded by the `read_cell_library` command.

Usage

```
write_cell_library new_lib_filename [-replace]
    [-INCLUDE_CREATED_dft_cell_selections | -ONLY_DFT_cell_selections ]
```

Description

Writes out a single library file of all library files loaded by the `read_cell_library` command, or alternately (if not using Tessent Shell) loaded at tool invocation with the `-library` switch; this single library file can subsequently be loaded by the `read_cell_library` command or at tool invocation with the `-library` switch to reproduce the internal library state that would exist if the same set of original files was loaded.

Note

 Library cells that were read into the tool using the `read_verilog` command with associated Verilog compiler directives like ``celldesign` and ``endcelldesign` are not included in the output of the `write_cell_library` command.

A `dft_cell_selection` wrapper is optional information that can be in the same file as the tessent cell library models, or in a separate file. A `dft_cell_selection` wrapper allows you to specify the cells (which are also models in the cell library) to use for DFT test logic applications such as scan insertion. Also, the `get_dft_cell` command accesses information from the `dft_cell_selection`. For example, if you need a 2-input AND cell, and one is specified in the `dft_cell_selection` wrapper, then that referenced model is used when you need such a cell for the test logic being inserted into your netlist, or when the “`get_dft_cell` and `--inputs 2`” command is issued. For more information, refer to the `get_dft_cell` command description in this manual.

If no `dft_cell_selection` wrapper is provided, a default will be created when a command is issued which causes elaboration, such as `get_dft_cell` or `set_current_design`, and will be given the name “`default`”. A model will be selected with an appropriate learned `simulation_function` (“`and`” in the example described above), and an attempt is made to determine which of those models (if any) is best. Note that there is no need to provide the `simulation_function` which is written out for informational purposes when you issue `write_cell_library`, as it is discarded when the file is parsed. Models with a `cell_type` specified in the model in the tessent cell library file (such as “`cell_type = and`”) are preferred over models with no `cell_type` specified when the file is parsed. Also, any model already instantiated in the current design netlist is preferred over a model which has no instances in the current design.

If both the default `dft_cell_selection` wrapper and models are written out, the learned `cell_type` is also written in appropriate models. These should also be checked and corrected if the file is to

be read back in for use in subsequent runs of some test tool requiring a tessent cell library. You can find these cells by searching for the string “// Learned cell_type”, as illustrated next.

Note

 All parsed input is considered to be user input, regardless of comments or names. If you do not change the default dft_cell_selection name from “default”, then that is the name for your selected dft cells. If you do not check and edit the default dft_cell_selection wrapper, but instead read it back as written, all the learned cell_type will be considered as user-provided and approved cell_type, not learned. This is true despite any comment in the parsed library file saying a cell_type was learned – that is ignored because when parsed, the cell_type is now user-specified, not learned. These cells will no longer defer to other cells whose cell_type was originally specified in the library models.

The following examples illustrate the output resulting from a model having its cell_type learned rather than specified when the model was parsed, combined with -include_default_dft_cell_selection. Note that the simulation_function is always learned and is always written out, but it is informative only and is discarded and recalculated each time the file is parsed. A cell_type must have a compatible simulation_function (for example, simulation_function should be mux for cell_type mux or cell_type clock_mux) to be used for test logic, even if you set the cell_type.

If the -include_default_dft_cell_selection switch is specified, and a default dft_cell_selection wrapper exists:

```
model TC4MUX2XC
  (Y, D0, D1, S0)
(
  cell_type = mux; // Learned cell_type
  simulation_function = mux;
  ....
```

If the -include_default_dft_cell_selection switch is not specified, or a default dft_cell_selection wrapper does not exist:

```
model TC4MUX2XC
  (Y, D0, D1, S0)
(
  simulation_function = mux;
```

Arguments

- *new_lib_filename*

A required string that specifies the location of the file to which the library contents are written in cell/model format. If this file is subsequently loaded using “read_cell_library new_lib_filename” or alternately (if not using tessent shell) using “-library new_lib_filename” at invocation, the internal library state that existed when the file was written will be effectively restored.

- **-replace**
An optional switch that specifies to replace the contents of the file specified by the new_lib_filename argument if the file already exists.
- **-INCLUDE_CREATED_dft_cell_selections**
An optional switch that writes out the created dft_cell_selection information. If no dft_cell_selection is parsed, the default dft_cell_selection will be written at the top of the tesson cell library file, before the models are written. Otherwise, the parsed dft_cell_selections will be written including the augmented dft_cell_selection information. Models with a learned cell_type will also have the cell_type written in the model. If this switch is not specified, only user specified cell_type are written in the model.
- **-ONLY_DFT_cell_selections**
An optional switch that allows you to write out only the dft_cell_selections to a file. If no parsed dft_cell_selection exists, in which case a default dft_cell_selection will be written, it is recommended that you change the name from “default” and adjust the default dft_cell_selection to reference the cells that you prefer, and then use the result in future tool invocations to achieve deterministic selection of the cells used for test logic. The file containing the edited dft_cell_selection wrapper can then be given to the read_cell_library command as a tesson cell library filename.

Examples

Example 1

The following example assumes the tool is invoked with the -library switch and the library files are loaded. The write_cell_library command writes the cells/models contained in those files to the new library file new_library.tcelllib. If the new_library.tcelllib file already exists the file is not written.

```
$ fastscan -library mylib.tcelllib
SETUP> write_cell_library new_library.tcelllib
```

Example 2

The following example assumes the tool is invoked without the -library switch and the library files are loaded after invocation with the read_cell_library command. The write_cell_library command writes the cells/models contained in those files to the new library file new_library.tcelllib. If the new_library.tcelllib file already exists, its contents are replaced with the new library data.

```
$ tessent -shell
SETUP> read_cell_library pad_lib.tcelllib
SETUP> read_cell_library standard_lib.tcelllib
SETUP> read_cell_library my_lib.tcelllib
SETUP> write_cell_library new_library.tcelllib -rep
```

Related Topics

[delete_cell_library](#)

[read_cell_library](#)

[write_config_data](#)

Context: unspecified, all contexts

Mode: all modes

Writes the configuration data presently in memory into a file. When using the -wrappers option, the data written to the file can be limited to some specific wrappers.

Usage

```
write_config_data filename [-wrappers wrapper_spec] [-specify_all] [-indent integer]  
[-replace] [-original_hierarchy] [-no_banner]
```

Description

Writes the configuration data presently in memory into a file. When using the -wrappers option, the data written to the file can be limited to some specific wrappers.

Arguments

- ***filename***

A required string option that specifies the name of the file into which to write the configuration data.

- **-wrappers *wrapper_spec***

An optional switch and value pair that specifies a Tcl list of one or more names of configuration wrappers, or a collection containing one or many configuration wrapper objects. Only the specified wrapper elements are written to the file.

Just like with the [report_config_data](#) command, if *wrapper_spec* only contains a single wrapper of type CSV, the format of the file will be changed such that a true CSV file is written out and ready to be read in to tools that support the CSV format such as Excel. This behavior is disabled if you use the -original_hierarchy option. A true CSV file has no wrapper and the line termination character is the carriage return instead of the semicolon, as is the case in a normal configuration data file.

- **-specify_all**

An optional switch that specifies that unspecified properties and singular wrappers all show up in the written file with the unspecified properties specified to their default values. By default, unspecified properties and singular wrappers do not appear in the output file.

- **-indent *integer***

An optional switch and value pair that specifies to change the amount of white spaces used for indentation. By default, sub-elements are indented by two characters with respect to their parent wrapper.

- **-replace**

An optional Boolean switch that suppresses the error that normally is issued if the filename already exists. When the -replace option is used, the tool silently deletes the existing file before creating the new one.

- **-original_hierarchy**

By default, the wrappers specified with the -wrappers options become root wrappers in the output file. You can use this option to preserve their original hierarchy and include all parent wrappers above them. When you write out a sub-wrapper without preserving the original hierarchy, you can then read it back into any valid parent wrapper using the “[report_config_data -in_wrapper](#)” command and option.

- **-no_banner**

An optional Boolean option that suppresses the banner that is normally written at the top of the file. The banner format is the following:

```
-----  
// File created by: Tesseract Shell  
// Version: 2014.1  
// Created on: Mon Jan 20 12:12:28 PST 2014  
-----
```

Notice that the banner is also removed when the file is output in the true CSV format. See the description of the -wrappers option for more info about the CSV format.

Examples

The following example shows the effect of all options.

```
#shows the configuration data currently in memory  
report_config_data  
  
tmp(1) {  
    subWrapper1 {  
        prop1 : v1;  
    }  
    subWrapper2 {  
        prop1 : v1;  
    }  
}  
  
# Standard write to a file  
write_config_data file1  
cat file1
```

```

//-----
// File created by: Tessent Shell
// Version: 2014.1-prerelease
// Created on: Mon Jan 20 12:47:09 PST 2014
//-----
tmp(1) {
    subWrapper1 {
        prop1 : v1;
    }
    subWrapper2 {
        prop1 : v1;
    }
}

# Limit write to one sub wrappers
write_config_data file2 -wrappers tmp(1)/subWrapper1
cat file2

//-----
// File created by: Tessent Shell
// Version: 2014.1-prerelease
// Created on: Mon Jan 20 12:47:09 PST 2014
//-----
subWrapper1 {
    prop1 : v1;
}

# Preserve the original hierarchy
write_config_data file3 -wrappers tmp(1)/subWrapper1 -original_hierarchy
cat file3

//-----
// File created by: Tessent Shell
// Version: 2014.1-prerelease
// Created on: Mon Jan 20 12:47:09 PST 2014
//-----
tmp(1) {
    subWrapper1 {
        prop1 : v1;
    }
}

#Remove the banner. Show error when file exist and -replace is not used.
catch {write_config_data file3 -no_banner}

// Error: File file3 cannot be overwritten unless you use the option -replace.

write_config_data file3 -no_banner -replace
cat file3

tmp(1) {
    subWrapper1 {
        prop1 : v1;
    }
    subWrapper2 {
        prop1 : v1;
    }
}

```

Related Topics

[report_config_data](#)

[write_core_description](#)

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: Retargetable pattern generation is enabled in patterns -scan_retargeting context ([set_current_mode](#) -type internal).

Writes out the core description corresponding to the current chip level.

Usage

`write_core_description file_name [-replace]`

Description

Writes out the core description corresponding to the current design.

The tool only writes out the current mode of the current core. It also writes out the core descriptions of any of its core instances so that the definition of the current core's current mode is self-contained.

The core description file can be generated in the following cases:

- After ATPG for the internal mode of a core, so that information can be fed forward for pattern retargeting.
- For the top level during pattern retargeting, so it can be used for reverse-mapping of chip-level failures back to the core level for diagnosis.
- For logicBIST in the patterns -ijtag context when generating chip-level patterns.

This command works in the -scan_retargeting context because this file is required as input for failure mapping.

Hybrid TK/LBIST Flow Specifics

Writes the description of the core for the logicBIST mode of operation. This includes description of the core pins, scan chain information, and EDT/logicBIST hardware registers (essentially, all information to capture the core-level logicBIST mode operation). Refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

Arguments

- *file_name*

A required switch that specifies the pathname of the file to which the core description is written.

- *-replace*

An optional switch that specifies to overwrite the existing file *file_name*.

Examples

The “[Retargeting Example](#)” in the *Tessent Scan and ATPG User’s Manual* demonstrates the use of this command in a design context.

Related Topics

[add_core_instances](#)
[delete_core_instances](#)
[read_core_descriptions](#)
[report_core_descriptions](#)
[report_core_instances](#)

write_core_timing_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.

Usage

`write_core_timing_constraints filename_prefix [-Replace]`

Description

Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.

If launch-off capture (broadside) transition patterns are used, fast capture constraints are generated, otherwise slow capture constraints are generated.

For more information on the timing files, see “[SDC Timing File Generation](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- ***filename_prefix***
A required string that specifies the naming prefix for the timing files.
- **-Replace**
An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example outputs timing files for the scan path and ATPG settings in the core design, using *atpgm* as the filename prefix:

```
write_core_timing_constraints atpgm
// Writing atpgm_core_shift_sdc.tcl
// Writing atpgm_slow_capture_sdc.tcl
```

Related Topics

[write_edt_files](#)

write_design

Context: all contexts

Mode: all modes

Writes the current design in Verilog netlist format to the specified file.

Usage

Usage 1: Creating interface modules

```
write_design -output_file filename [-banner banner] [-modules modules_list]
    [-interface] [-verbose] [-replace]
```

Usage 2: Writing out created modules in rtl context

```
write_design -output_file filename [-banner banner] [-created | -modules modules_list]
    [-hierarchical]] [-exclude_modules exclude_modules] [-verbose] [-replace]
```

Usage 3: Writing out read-in modules in rtl context

```
write_design {-output_directory directory | -original_location [-extension_suffix suffix]}
    [-modified | -all_read | -modules modules_list [-hierarchical]]
    [-exclude_modules exclude_modules] [-write_out_updated_f_files] [-verbose]
    [-file_map file_map_var] [-replace]
```

Usage 4: Writing out read-in or created modules in no_rtl context

```
write_design -output_file filename [-banner banner]
    [-modified | -created | -modules modules_list
    [-hierarchical]] [-exclude_modules exclude_modules] [-verbose] [-replace]
```

Usage 5: Writing out the graybox view of the current design

```
write_design -output_file filename [-banner banner] [-graybox] [-verbose] [-replace]
```

Usage 6: Writing out the modified design in the dft_inserted_designs directory of the TSDB

```
write_design -tsdb [-verbose]
```

Usage 7: Writing out the graybox view of the current design in the dft_inserted_designs directory

```
write_design -tsdb -graybox [-verbose]
```

Usage 8: Creating a dft_inserted_designs directory in the TSDB with a softlink to the read-in concatenated netlist

```
write_design -tsdb -softlink_netlist [-verbose]
```

Description

Writes the current design in Verilog netlist format to the specified file.

Use the -modules switch to write only specific modules. The default is to write the top module with all the modules in the hierarchy below except the created modules and the modules below them. The file organization of the read in files is never changed when using the -rtl mode of the [set_context](#) command. See the section “[Difference Between -rtl and -no_rtl Mode](#)” for the

difference between `-rtl` and `-no_rtl` mode when it comes times to write out the design using the `write_design` command.

Note

 You cannot use the `write_design` command to overwrite files containing RTL modules that were read in with the `read_verilog` command. See the section “[Difference Between -rtl and -no_rtl Mode](#)” below for an explanation why.

Note

 The `write_design` command is disabled if you read in a flat model using the `read_flat_model` command.

Note

 Verilog modules originally defined inside `'celldefine` and `'endcelldefine` directives are not included in the output netlist. For more information about using `'celldefine` to define cell boundaries, refer to the “[Fault Locations](#)” discussion in the *Tessent Scan and ATPG User’s Manual*.

Usage 1 — This usage case writes out the interface definition of the specified modules only. The port and parameter definitions are saved but everything else is deleted from the module; this has no impact on the module definition in memory and only affects the content of the file being written out. If you do not specify the `-modules` option, the interface of the current design is written out.

Usage 2 — This usage case writes out the created modules in the `rtl` context. Created modules are modules created using the `create_module` command; these modules do not have a source file associated with them so you must specify one using the `-output_file` option. You can write all created modules into a single file using the `-created` option; this only works if all of your created modules have a common language because modules from different languages (VHDL versus Verilog) must be written into different files.

If you do not specify either `-modules` or `-created` options, the tool infers the following: “`-modules [get_current_design] -hierarchical`.” In this case, the tool generates an error if the current design is a created module and you did not specify the `-output_file` switch. When you specify the `-modules` option, you can only specify created modules; otherwise, an error is generated.

When you specify the `-hierarchical` option, read-in modules are automatically skipped.

Usage 3 — This usage case writes out read-in modules in the `rtl` context. Because RTL can contain many directives that affect how things are later compiled, the tool preserves the content and ordering of the RTL files when writing them out. For this reason, you can only write out read-in modules using either the `-output_directory` or the `-original_location` switch. The use of the `-output_file` option is not allowed for read-in modules in the `rtl` context.

By default, the tool does not write out the read-in -f files. You must specify the optional -write_out_updated_f_files switch to write out the -f files in addition to the read-in RTL files.

You can use the -all_read option to decompile and write all files previously read with read_verilog or read_vhdl, even if they contain no module definitions such as files containing only the definitions of System Verilog interfaces, `define statements, or files containing no modules instantiated below the current design.

You can use the -modified option to quickly write out only the modified modules, but if you also specify created modules, the tool does not write those out. However, you can write out the created modules using a separate write_design command invocation with the -output_file option specified.

If the -modules or -modified options are not specified, the tool infers the following: “-modules [get_current_design] -hierarchical.” In this case, the tool generates an error if the current design is a created module and you used the -output_directory or the -original_location switches. When you specify the -modules option, you can only specify read-in modules in this usage.

For more explanation about how read-in modules are written out in the RTL context, see the “[Difference Between -rtl and -no_rtl Mode](#)” section below.

Usage 4 — This usage case writes out read-in or created modules in the no_rtl context. In the no_rtl context, the design modules are written out with all compiler directives removed. The dependencies on the file order is thus removed, allowing the concatenation of any modules into common files. You can merge both read-in and created modules into the same file.

In the no_rtl context, the recommended use model is to write out the entire design hierarchically into a single file which is the default behavior when the -modules, -created or -modified option are not specified.

Usage 5 — This usage case writes out the graybox view of the current design into a single file. Only those instances and nets that have the in_graybox attribute set to true are kept in this file. All other nets and instances are removed. You can set this attribute using the [set_attribute_value](#) command but this attribute is normally populated using the [analyze_graybox](#) command. See Usage 7 below for how to get the graybox model saved into the TSDB.

Usage 6 — This usage case creates a [dft_inserted_designs](#) directory with the modified design that you created using a custom script. If ICL is elaborated, it is written out to the new dft_inserted_designs directory with the [tessent_design_id](#) attribute updated to match the new *design_id* define with the [set_context](#) command. The concatenated PDL file is also copied and the .tcd file and the .tsdb_info files are also created. You use this case when you want to run a custom editing script. Saving it using “write_design -tsdb” will store it inside the TSDB such that you can read it back later in using the [read_design](#) command.

Usage 7 — This usage case writes out the graybox view in the `dft_inserted_designs` directory. When using the `-graybox` switch, an error is generated if the `analyze_graybox` command was not called prior. The graybox view can be saved in a `dft_inserted_designs` directory that was previously created using the “`write_design -tsdb`” or “`insert_test_logic -write_in_tsdb on`” command. If the current “`set_context -design_identifier`” value does not match an existing `dft_inserted_designs` directory, a new one is created with all the associated files: the ICL, PDL, TCD, and the TSDB info files.

Usage 8 — The `-softlink_netlist` usage case is only usable in the `-no_rtl` mode right after the design was elaborated in the setup mode. It creates a `dft_inserted_designs` directory for the read-in netlist such that it can be easily loaded with `read_design` or by the `run_testbench_simulations` commands. This option is typically used to create a TSDB entry for the post synthesis netlist when scan insertion is not done in Tessent Shell. The post synthesis TSDB view is created by “`insert_test_logic -write_in_tsdb on`” when the scan insertion is done in Tessent Shell. It is also used to create a TSDB entry for the post layout netlist such that the `read_design` command can be used in ATPG to load this netlist, create a graybox view when it is a wrapped core and automate the design loading in `run_testbench_simulations`.

Difference Between `-rtl` and `-no_rtl` Mode

There are many differences that exist between `rtl` and `netlist` files. Those differences are listed and contrasted in [Table 6-23](#). Because of the characteristics of the `rtl` files mentioned in the table, the module organization of `RTL` files cannot be changed.

[Figure 6-9](#) shows the differences that exist inside the tool when reading `rtl` vs `netlist` files. In both cases, the data model inside the tool only consists of the structural elements found inside the modules, instances, ports, pins, and nets. The different color used for the input files of the `rtl` mode are meant to illustrate the various languages and formats the files may have in `rtl` mode. In the `netlist` mode, the files are always with language Verilog and format 2001. When it comes time to write out the modified `netlist`, all the information is in the data model and a new `netlist` is reconstructed from scratch using the information inside the tool. In the `rtl` mode, only the edits made to the structural objects (module, instances, pin, port, net) are present in the internal data model. The effects on the `rtl` files are only reflected when the `write_design` command is used. The original files containing the modified modules are used and the edits are patched in a surgical fashion such that all white spaces, comments and carriage returns in the original files are preserved. Only the pins, ports, and nets affected by the edits show modifications. Because each file came in with a unique language and format, all written out files preserve the unique language and format as illustrated with the coloring of the files. [Figure 6-10](#) shows the effect of doing two insertion passes on the same design. Some files were never modified. Some files were only modified by the first insertion pass. Some files were only modified by the second insertion pass. Finally, some files were modified by both insertion passes. The design view that represents the design as it exists after the second insertion pass has files located in the original location, in the `dft_inserted_designs` directory of the first insertion pass and in the `dft_inserted_designs` directory of the second insertion pass. Refer to the `read_design`, `run_testbench_simulations` and the `write_design_import_script` command descriptions to see how the tool provides powerful commands to help reload the edited `rtl` design files.

Table 6-23. Difference between rtl and Netlist files

RTL	Netlist
9 distinct formats, no superset parser Verilog (1995 2001 sv31a sv2005 sv2009) VHDL (1987 1993 2002 2008)	1 format Verilog 2001
Content of read-in file affects the files read afterward. Ex :`define, packages, SV definitions,...	Read order has no effect
File structure and read order must be preserved one format per file read order cannot be changed	Support concatenating all modules into one file
Typically one file per module or functional block manually edited and maintained revision control system like CVS or Git	One file per physical block
One module name can have multiple implementations parameters, generate blocks, logical libraries, ...	One module name, one implementation

Figure 6-9. Writing Out the Modified Design in rtl vs no rtl Modes

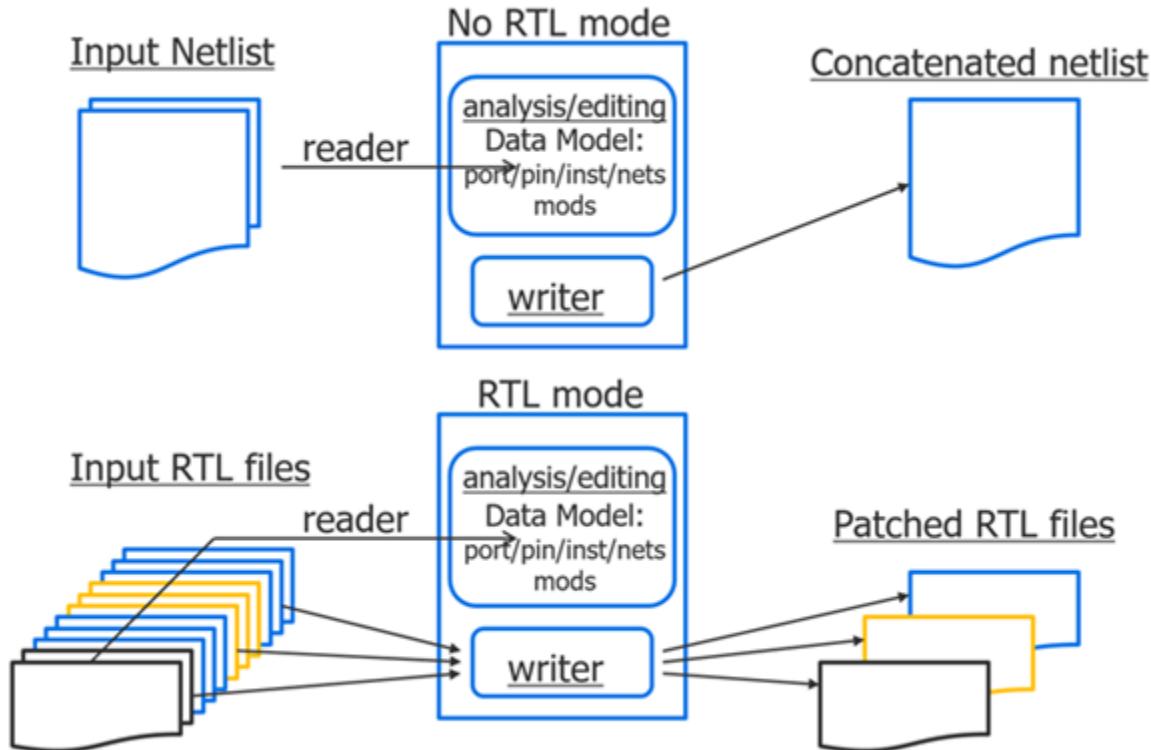
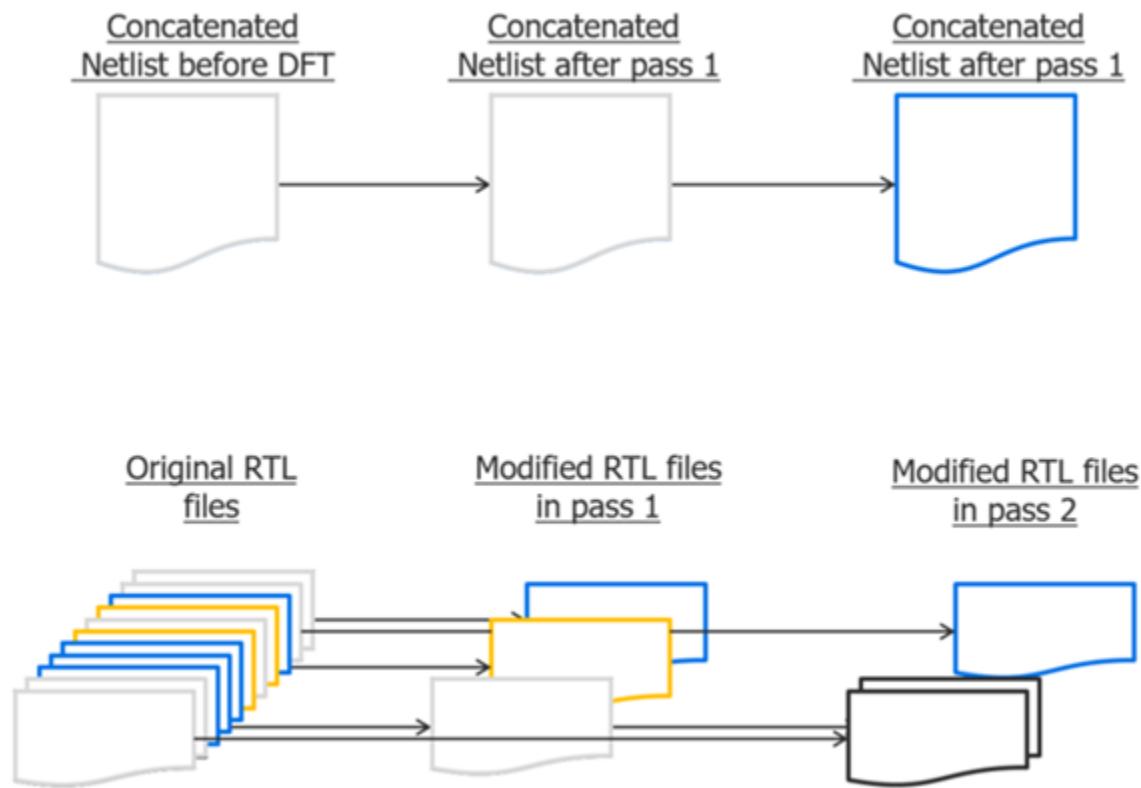


Figure 6-10. Effect of Successive Design Editing Passes on the Design View



Arguments

- **-output_file *filename***

A switch and string pair that specifies the name of the file in which to write all specified modules. This switch can only be used in the rtl context when writing created modules as described in Usage 2 above. It can be used for any modules in the no_rtl context.

- **-banner *banner***

An optional switch and string pair that specifies a header comment to be written as the first line of the output file.

- **-output_directory *directory***

A switch and string pair that specifies the directory in which to write all specified modules. All files containing at least one of the specified modules will be written out. Subdirectories within the specified output directory will be created if the read-in files were stored in separate directories in order to avoid file name collisions. This switch is only allowed in -rtl mode and defines Usage 3 described above. See the “[Difference Between -rtl and -no_rtl Mode](#)” section for an explanation why the file structure is not modified when using the rtl context.

- **-original_location**

A switch that specifies to write out the specified modules in the same directory that they were read in from. All files containing at least one of the specified modules in the -modules

switch value will be written out. The file name and directory location of the source file will be used except that the default or specified -extension_suffix value will be appended to the file extension.

This option preserves the same file and directory structure as the original rtl description. The modified rtl files can be loaded into the next tool using the original -y options by simply modifying the extension search list. You must, however, have write access in the source directories. The source file can be write protected but not the source directory.

The commands [process_dft_specification](#), “[insert_test_logic -write_in_tsdb on](#)”, and “[write_design -tsdb](#)” all create a “[dft_inserted_designs](#)” TSDB container for the new design view. The modified rtl files are stored inside that directory and the [.design_source_dictionary](#) file contains the information to load that design view within the [read_design](#), [write_design_import_script](#), and the [run_testbench_simulations](#) commands. If you want the modified rtl files to be stored back beside the original file locations as it was done in the LV Flow, then issue the following command before leaving the insertion mode (for example, after having called the [process_dft_specification](#) command but before calling [extract_icl](#)):

```
write_design -modified -original_location -extension_suffix _et
```

The file dictionary will be written into the TSDB when exiting the insertion pointing to the modified rtl files written back into their original location.

- **-modules *module_list***

An optional switch and value pair that specifies the names of the modules to write out. The *module_list* value is a Tcl list of module names, or a collection of modules as returned by the [get_modules](#) command.

In the rtl context when using multiple logical libraries and vhdl architectures, use the [get_modules](#) commands to select modules from a specific library or architecture. A module name in the *module_list* is only looked for in the default library with the default architecture.

In the rtl context, you cannot use the -output_file switch when writing out read-in modules. The written out file always contain the exact same modules list that existed in the read-in file. If you use the -modules switch to refer to a module, the source file that contains the read-in module will be written out containing all modules that existed when it was read-in. See the “[Difference Between -rtl and -no_rtl Mode](#)” section for an explanation of why the file structure cannot be modified for read-in files in rtl mode.

- **-hierarchical**

An optional switch that writes out all child modules below the specified modules. This switch can only be used with the -modules switch. If the -modules switch includes created modules, read-in modules found below those modules are ignored. If the -modules switch includes read-in modules, created modules found below those modules are ignored. An error is generated if the -modules switch contains both read-in and created modules as those consist of different usage. See Usage 2 and 3 above for an explanation of the difference. See the “[Difference Between -rtl and -no_rtl Mode](#)” section for more details.

- **-exclude_modules** *exclude_modules*

An optional switch and value pair that excludes modules from being written out. This argument is useful only when the -hierarchical option is inferred or specified. All modules below these excluded modules are also excluded if they are not specified in the -modules list or found below other non-excluded modules.

In the rtl context, when using multiple logical libraries and vhdl architectures, use the [get_modules](#) command to select modules from specific libraries or architectures. A module name in the module_list is only looked for in the default library with the default architecture.

In the rtl context, all modules found in a read-in file are always included in the written out file even if the -exclude_modules switch specifies to exclude some of those module. As soon as one module in the file is in the -module switch, the entire file is written out with all modules found in the read-in file. See the “[Difference Between -rtl and -no_rtl Mode](#)” section for more details.

- **-write_out_updated_f_files**

An optional switch to add the read-in -f files to the written rtl files. The -f files are modified to point to the new location of the written directories and files. This may change the directory structure of the written files if the -f files were located higher in the directory tree as the rtl files.

- **-all_read**

An optional switch that decompiles and writes out all files previously read with [read_verilog](#) or [read_vhdl](#), even if they contain no modules. This switch is available only in the rtl context.

- **-created**

An optional switch that specifies to limit the written modules to only those *created*. Specifying this option is equivalent to using “-modules [[get_modules](#) -filter is_created].”

- **-tsdb**

An optional switch that specifies to write the design and all its associated files in the [Tessent Shell Data Base \(TSDB\)](#). See the Usage 6, 7, and 8 above for more details.

- **-graybox**

An optional switch and string pair that specifies that the content of the output files will not contain the complete module descriptions but only those nets and instances having the in_graybox attribute set to true. You can set this attribute using the [set_attribute_value](#) command but is normally populated using the [analyze_graybox](#) command.

This switch is available only in the no_rtl context.

- **-softlink_netlist**

An optional switch that is only usable in the -no_rtl mode right after the design was elaborated in the setup mode. It must be used with the -tsdb switch to create a [dft_inserted_designs](#) directory containing a softlink to the read-in netlist such that it can be

easily loaded with [read_design](#) or by the [run_testbench_simulations](#) command. For more information about this switch, see Usage 8 described above.

- **-interface**

An optional switch that specifies to write out only the interface of the current design or the interfaces of the modules specified by the **-modules** option.

- **-extension_suffix *suffix***

An optional switch and string pair that specifies the suffix to use to distinguish the written-out file names from the read-in file names when the **-original_location** option is used. The **-extension_suffix** defaults to “**_ts**.”

- **-verbose**

An optional switch that specifies to print the name of each written file, one per line.

- **-file_map *file_map_var***

An optional switch and string pair that creates a list of files the tool modifies during the design writing operation. Use this switch when doing flows that require the generation and use of context views for designs requiring uniquification. The *file_map_var* will show which files got modified, enabling you to update your design load script.

- **-replace**

An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

Examples

Example 1

The following example writes out the current design and every other design module instantiated below it into a common file.

```
write_design -output_file design_out.v
```

Example 2

The following example generates an interface module description for the current design inside the file called mod.shell.

```
write_design -output_file mod.shell -interface
```

Example 3

The following example writes out all modules within the current design but stops on CoreA and CoreB. Separate commands are used to write out all modules within CoreA and CoreB. Notice how the first command does not specify the **-modules** option which means the current design is to be written out and the **-hierarchical** switch is inferred.

```
write_design -output_file myTop.v -exclude_modules {CoreA CoreB}
write_design -output_file CoreA.v -modules CoreA -hierarchical
write_design -output_file CoreB.v -modules CoreB -hierarchical
```

Example 4

The following example uses the `-no_hdl` option of the `read_design` command to load the information about `my_design` from the pre-synthesis TSDB directory and combine it with the post-synthesis netlist to create a complete gate level TSDB directory. This will later allow using “`read_design my_design -design gate`” to create patterns.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Refer to [Example 2 in run_testbench_simulations](#) for a usage example about how to use the `write_design -tsdb -softlink_netlist` command to create a TSDB `dft_inserted_designs` container for a netlist generated from synthesis or layout, and on which you want to use the `run_testbench_simulations` command to simulate the patterns described in your PatternsSpecification.

Related Topics

[delete_design](#)
[set_current_design](#)
[read_verilog](#)
[read_vhdl](#)

write_design_import_script

Context: dft, patterns

Mode: setup, analysis

Generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted.

Usage

```
write_design_import_script [file_name] [-replace] [-module module_spec]
[-design_id design_id] [-use_relative_path_to directory]
[-exclude_sub_blocks]
```

Description

This command generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted. The script analyzes all the source files that were read into Tessent Shell and reported into the *design_name.design_source_dictionary* for the current design that is generated into the [Tessent Shell Data Base \(TSDB\)](#). By default, all the sub-blocks of the design are also analyzed by the script, unless the option `-exclude_sub_blocks` is specified. When the *design_name.design_source_dictionary* has not been generated for the current design, the generated script uses the information in memory about the source files that were read into Tessent Shell when the command is issued.

Arguments

- *file_name*

An optional string that specifies the script file name. When *file_name* is not specified, the name defaults to:

design_name.tool_name_import_script

- `-replace`

An optional switch that specifies overwriting *file_name* when the file exists.

- `-module module_spec`

An optional switch and string pair that specifies a specific module name in an opened TSDB that has been DFT inserted. The default is to use the current design and current design identifier.

- `-design_id design_id`

An optional switch and value pair that specifies a specific *design_id* when a module has been DFT inserted more than once.

- `-use_relative_path_to directory`

An optional switch and value pair that specifies a directory from which the generated script is processed. The files being referenced are relative to this directory. By default, the generated script includes the full paths to the files to process.

- [-exclude_sub_blocks](#)

An optional switch that specifies to ignore the analysis of the sub-blocks instantiated in the main module in the generated script. By default and if this switch is not specified, the generated script considers the design source dictionary for the selected DFT inserted design and all the sub-blocks that are included in the selected design. The inclusion stops and does not include any module that is a physical block. All the TSDBs used to assemble the selected module must be opened in order for the sub-block information to be accessible.

Examples

Example 1

The following produces the report for the current design in memory, with full paths and including all the sub-blocks. The resulting script is displayed when in interactive mode and is reported in the Tesson Shell log file.

[write_design_import_script](#)

Example 2

The following produces the report for the design *my_core* with paths relative to the subdirectory *synth/my_core* and without the sub-blocks. The generated script is saved in the file *synth/my_core/analyze.tcl*.

```
file mkdir synth/my_core
write_design_import_script analyze.tcl -replace -module my_core \
    -exclude_sub_blocks -use_relative_path_to ./synth/my_core
```

Related Topics

[run_synthesis](#)

write_design_source_dictionary

Context: dft, patterns

Mode: setup, analysis, insertion

Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location.

Usage

`write_design_source_dictionary output_file [-dictionary dictionary] [-replace]`

Description

Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location. You can subsequently source this written-out design-source dictionary to reload the design using the “[read_design -from_dictionary \\$design_source_dictionary](#)” command. Refer to the [example](#) to see how you use this command to package an RTL test case into a single directory. If you omit the -dictionary switch, then the design-source dictionary returned by the “[get_design_sources -file_dictionary](#)” command is written out. You can also choose to store the design-source dictionary returned by the “[get_design_sources -file_dictionary](#)” command into a variable, make edits to the dictionary stored into the variable using the Tcl “dict” command, and then write out the modified dictionary using the “-dictionary \$var” switch, where “var” is the Tcl variable that holds the dictionary you modified.

Note

 If you want to write out a design-source dictionary that reflects the state of the RTL design after you have edited the design in insertion mode, then you must first go to setup mode to synchronize the design in order for the file dictionary to reflect the files with your edits.

Arguments

- ***output_file***

A required string argument that specifies the name of the file into which the tool will write the design-source dictionary. All of the file and directory paths contained in the design-source dictionary file are relative to the location of the output file.

- **-dictionary *dictionary***

An optional switch-string pair that specifies a Tcl dictionary the tool will write out. Typically, you do not use this switch. When unspecified, the tool writes the design-source dictionary that is returned by the “[get_design_sources -file_dictionary](#)” command. You can choose to store the design-source dictionary returned by the “[get_design_sources -file_dictionary](#)” command into a variable, make edits to the dictionary stored into the variable using the Tcl “dict” command, and then write out the modified dictionary using the -dictionary \$var switch, where “var” is the Tcl variable that holds the dictionary you modified.

- **-replace**

An optional Boolean switch that suppresses the error the tool normally issues when the specified ***output_file*** already exists. Instead of issuing an error, the tool silently replaces the file.

Examples

The following example demonstrates writing out the complete RTL source files into a single directory with an associated design-source dictionary.

You can subsequently tar the dictionary file and the RTL directory as a test case. When the dictionary file and the RTL directory are untarred at the destination, the file dictionary is sourced into the tool using the **read_design -from_dictionary \$design_source_dictionary** command, which reads the design from the RTL directory. Use the **-skip_existence_checks** switch to suppress the error that the tool issues if the design you are loading uses the **-y** and/or **-v** libraries, or **+incdir** specifications. The design-source dictionary will contain paths to the original path as well as the new path found within the RTL directory:

```
# To store the complete RTL file, use those commands:  
set current_design_name [get_single_name [get_current_design]]  
write_design -output_directory ${current_design_name}_rtl_files -all_read  
write_design_source_dictionary ${current_design_name}.design_source_dictionary  
  
# Tar up the file and the directory and untar it at the destination  
# To reload the design, use those commands:  
set current_design_name my_design  
source ${current_design_name}.design_source_dictionary  
read_design -from_dictionary $design_source_dictionary -skip_existence_checks  
set_current_design $current_design_name
```

write_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.

Usage

```
write_diagnosis [-SHORT] [-ENCoded] [-FORmat {TEXT | LAYout_marker | CSV}...] [-FILE name [-REPlace]] [-XMAP {V2LVS | CELL_HIERarchy}...]
```

Description

Writes diagnosis reports in text, comma-separated values (CSV), and layout (physical) marker location formats.

If you specify write_diagnosis with no arguments, Tessent Diagnosis creates an ASCII text diagnosis report and names the file using the *failure_filename* from the diagnose_failures command and puts the results file where *failure_filename* is located.

Arguments

- **-SHORT**

An optional switch that changes the diagnosis report from the layout-aware diagnosis report format to the traditional diagnosis report. Allows for backward compatibility with other tools for reading in the diagnosis report.

Using this option does not change the computed diagnosis result including layout-aware diagnosis information, but rather suppresses the display of the layout-aware information.

Note



This option performs the layout-aware diagnosis, but suppresses the display of this information.

- **-ENCoded**

An optional switch specifying encoding a suspect's pin pathname, net name, and cell name in the diagnosis output file.

- **-FORmat {TEXT | LAYout_marker | CSV}**

An optional switch and single or multiple literals specifying the report's output format. By default, the write_diagnosis command creates an ASCII text version of the report if you specify no arguments.

Note



To produce the location tables (XMAP tables), you must select the default ASCII text report format (-format TEXT) for the write_diagnosis command.

If you use the -FORmat switch, then you must specify at least one of the following options:

- TEXT — Specifies ASCII text format.
- LAYout_marker — Specifies layout coordinate format.
- CSV — Specifies comma separated value (CSV) format.

You can specify any combination of these literals. For example, the following syntax creates both text (*example.diag*) and CSV (*example.csv*) reports:

```
write_diagnosis -format text csv -file example
```

- **-FILE *name***

An optional switch and string pair specifying the filename (*name*) of the report. The tool uses the *name* value for the report filename, regardless of format, and identifies the format using the following filename suffixes:

- *name.csv* — CSV format diagnosis report.
- *name.diag* — ASCII text format diagnosis report.
- *name.lay* — Layout markers.
- *name.enc.csv* — Encoded CSV diagnosis report.
- *name.enc.diag* — Encoded ASCII text diagnosis report.
- *name.enc.lay* — Encoded layout markers.

- **-REPlace**

An optional switch for overwriting an existing file.

- **-XMAP {V2LVS | CELL_HIERarchy}**

An optional switch and single or multiple literals specifying the outputting of SPICE-mapped nets, Verilog module hierarchy, or both in the diagnosis report.

If you use the -XMAP switch, then you must specify at least one of the following options:

- V2LVS — Specifies outputting the SPICE-mapped nets.
- CELL_HIERarchy — Specifies outputting Verilog module hierarchy.

You can specify any combination of these switches. For example, the following syntax creates both the SPICE-mapped nets and the Verilog module hierarchy in the diagnosis report:

```
write_diagnosis -XMAP V2LVS CELL_HIERarchy
```

Examples

Example 1

The following example creates a CSV and ASCII text diagnosis report with the filename *foo* in the *bar* directory, which must exist:

```
write_diagnosis -format text csv -file bar/foo
```

If you list the *bar* directory, then the following information is displayed:

```
% ls -l bar  
...  
-rwxrwxr-x    1 user account 1339 Aug 10 09:53 foo.csv  
-rwxrwxr-x    1 user account 1768 Aug 10 09:53 foo.diag
```

Example 2

The following example creates encoded CSV, ASCII text, and layout markers with the filename *foo* in the *bar* directory, which must exist:

```
write_diagnosis -encoded -format text csv layout_marker -file bar/foo -replace
```

If you list the *bar* directory, the following information is displays:

```
% ls -l bar  
...  
-rwxrwxr-x    1 user account 1241 Aug 10 09:53 foo.enc.csv  
-rwxrwxr-x    1 user account 1502 Aug 10 09:53 foo.enc.diag  
-rwxrwxr-x    1 user account 40495 Aug 10 09:53 foo.enc.lay
```

Related Topics

[report_diagnosis](#)

[diagnose_failures](#)

write_edt_files

Context: dft -edt

Mode: analysis

Creates the files that implement the EDT logic.

Usage

```
write_edt_files filename_prefix [-Replace]
  [-VErilog | -VHdl [-Signal_type STD_Logic | STD_Ulogic]]
  [-SYnthesis_script {DC_Tcl | DC_Shell | RTL_Compiler}]
  [-NO_Timing_constraints | -Timing_constraints]
  [-Nortl_prefix | -RTI_prefix prefix_string]
  [-REDuced_netlist]
  [-Insertion Ts | Dc]
  [-Single_module_per_file]
  [-UNIque_bypass_lockup_clock]
  [-EXClude_modules list_of_modules]
  [-NO_SYNOpssyse_pragma]
  [-NO_WRITE_Design]
  [-IJtag {Off | data_ports}]
  [-TSdb [-DESIGN_Id design_id]]
```

Description

Creates the files that implement the EDT logic.

For more information, see “[Creation of the EDT Logic](#)” in the *Tessent TestKompress User’s Manual*. By default, if no other switches are specified, the following files are written:

- EDT logic RTL (**_edt.v*)
- Tessent Core Description (TCD) file (**_edt.tcd*)
- ICL file (*_edt.icl*)
- PDL file (*_edt.pdl*)
- Test procedure file (**_edt.testproc*)
- Dofile (**_edt.dofile*)
- Design Compiler (DC) synthesis script (**_dc_script.scr*)
- Bypass dofile (**_bypass.dofile*)
- Bypass test procedure file (**_bypass.testproc*)

The following additional files are written out for the external flow:

- Core top file (**_edt_top.v*)

-
- Core blackbox file (**_core_blackbox.v*)

If the [set_lpct_controller](#) command is enabled and issued, then the tool writes out the following additional file:

- LPCT TCD file (**_edt_lpct.tcd*)

When the EDT logic is configured for multiple compression configurations, a separate test procedure and dofile are written for each configuration. For more information, see “[Defining Dual Compression Configurations](#)” in the *Tessent TestKompress User’s Manual*.

By default, the language of the netlist read in is the format used for the EDT logic.

If you execute this command and you are using the internal IP location flow with Tessent Shell EDT insertion, the tool writes out all EDT files, inserts EDT IP into the design, and automatically changes the system mode from analysis to insertion.

If you execute this command and you are using the external IP location flow, the tool writes out all EDT files including the wrapper that instantiates the core design and EDT logic and remains in analysis mode. You can explicitly go to insertion mode, at which time you can further edit the core design and save the modifications. However, saving the design in insertion mode is only necessary if you have modified the design in insertion mode.

If you execute this command and you are using the internal IP location flow with DC-based EDT insertion, the tool writes out all EDT files and the DC insertion script and remains in analysis mode. If you change to insertion mode and modify the design, make sure the modifications are compatible with the DC synthesis script that was already generated.

Arguments

- *filename_prefix*

A required string that specifies a name prefix for the EDT logic files. The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the *filename_prefix*, and the files are written to that location. This option is mutually exclusive with the use of the -tsdb option.

- -Replace

An optional switch that replaces the contents of each file that already exists.

- -VERilog

An optional switch that writes the EDT logic in Verilog format.

- -VHdl

An optional switch that writes the EDT logic in VHDL 93 format.

- **-SYnthesis_script DC_Tcl | DC_Shell | RTL_Compiler**

An optional switch and set of literals that specify to generate a synthesis script:

DC_Tcl —

DC_Shell —

RTL_Compiler —

- **-SIGNAL_type STD_Logic | STD_Ulogic**

An optional switch and literal pair that specifies the type of signals and ports to use (std_logic or std_ulogic) when writing out the EDT logic in VHDL format.

Note

 When writing out the top-level wrapper (*_edt_top.vhd) and the black box description of the core (*_core_blackbox.vhd), the std_logic and std_ulogic declaration of signals and ports in the original netlist are preserved.

- **-REDuced_netlist (internal flow only)**

An optional switch that writes out the input netlist into two files as follows:

- <naming_prefix>_reduced_netlist.v (or .vhd) — Reduced netlist containing only the part of the original netlist required for making necessary connections between pad terminals and EDT logic pins during synthesis. Use this file instead of the original input netlist with the synthesis script. Then use the output netlist from the synthesis run along with the _rest_of_netlist.v file to generate test patterns.
- <naming_prefix>_rest_of_netlist.v (or .vhd) — Netlist file containing everything in the original netlist read into the tool except the modules written out in the _reduced_netlist.v file. Use the _rest_of_netlist.v file and the output netlist from the synthesis run to generate test patterns.

Use this switch to create a reduced input netlist to save time during synthesis. For more information, see “[Creation of a Reduced Netlist for Synthesis](#)” in the *Tessent TestKompress User’s Manual*.

Note

 This switch can only be used with the *-Insertion Dc* command.

- **-Single_module_per_file**

An optional switch that writes the EDT logic into multiple files with one Verilog module (or VHDL entity/architecture pair) in each file. Use this switch only if your synthesis flow requires the RTL files to be organized this way.

The number of files written depends on the EDT logic setup. When the EDT logic is located outside the core, the top-level wrapper file may be split up if any pins share channel outputs. The DC synthesis script is modified to read in each of the RTL files individually.

Note

-  If set up to use the internal flow (“set_edt_options -location internal”), the tool ignores the -Single_module_per_file switch for the gate-level core netlist when you include the -Reduced_netlist switch.

The -Single_module_per_file switch causes the tool to use a different naming convention for the output files as described in [Table 6-24](#).

Table 6-24. Single Module Per File Naming Convention

Old Name	New Name	Output File
<filename_prefix> _edt_top.v (or .vhd)	{<top_module_name> <rtl_prefix>} _edt_top.v (or .vhd)	Top-level wrapper (external flow only)
<filename_prefix> _edt.v (or .vhd)	{<top_module_name> <rtl_prefix>} _edt.v (or .vhd)	EDT circuitry
<filename_prefix> _core_blackbox.v (or .vhd)	<top_module_name> _blackbox.v (or .vhd)	Blackbox description of core (external flow only)

The additional *single module* files are named as follows:

- {<top_module_name> | <rtl_prefix>}_edt_<module_name>.v (or .vhd)
- {<top_module_name> | <rtl_prefix>}_edt_pinshare_logic.v (or .vhd)
(external flow only)

The RTL prefix string is used when constructing the preceding names only if you include the -Rtl_prefix switch; otherwise it uses the top-module name. If the command’s *filename_prefix* argument includes a pathname component, the tool still writes all files to that location even though the filename prefix may not appear in all filenames.

- **-Nrtl_prefix**

An optional switch that disables the default internal RTL naming feature. By default,

- **-RTl_prefix *prefix_string***

An optional switch and string pair that specifies a *prefix_string* to prepend to the module and instance names in the EDT logic RTL. The specified *prefix_string* must follow the normal naming rules for Verilog or VHDL identifiers.

- **-Insertion *Ts* | Dc (internal flow only)**

Optional switch and literal pair that determines the method used to insert the EDT logic inside the design core. Options include:

[Ts — Inserting EDT Logic During Synthesis](#)” in the *Tessent TestKompress User’s Manual*.

Note

 You must enter all commands for determining configuration, test coverage, and data volume before saving out the EDT logic files. After the composite netlist is written, . For more information, see “[ATPG Baseline Generation](#)” in the *Tessent TestKompress User’s Manual*.

Dc — Writes a Synopsys Design Compiler script to insert and synthesize the EDT logic inside the core netlist. For more information, see “[Synthesizing the EDT Logic](#)” in the *Tessent TestKompress User’s Manual*.

- **-NO_Timing_constraints | -Timing_constraints**

An optional switch that determines if SDC timing files are output. By default, no timing files are output. When you specify -timing_constraints, the tool reports both the fast capture and slow capture constraints. For more information, see “[SDC Timing File Generation](#)” in the *Tessent TestKompress User’s Manual*.

- **-UNIque_bypass_lockup_clock**

An optional switch that individually routes the clock pin of all scan cells involved with bypass lockup cells to the EDT logic. The clock pins created on the EDT logic boundary are named *edt_bypass_clk<n>* instead of top-level clock names.

- **-EXClude_modules *list_of_modules* (DC-based insertion only)**

An optional switch and string pair that specifies to exclude from the complete EDT logic-inserted netlist, those modules specified by the *list_of_modules* argument; *list_of_modules* and any sub-module used by one of the specified modules, unless the sub-module is also used by another module that is to be written out. The *list_of_modules* is a list of strings that are separated by spaces.

In the internal IP location flow with DC-based insertion, the tool writes out a Synopsys Design Compiler script to make EDT connections and to write out the modified design. Without the -EXClude_modules switch, the netlist is complete and contains a description of all the modules present in the input netlist. This switch excludes from the netlist the descriptions of the modules specified by the *list_of_modules* argument.

During pattern generation, you need to provide the complete view of the EDT-inserted design including the module descriptions you excluded from the IP-creation output netlist with this command.

Note

 If any of the modules excluded by this switch need to be modified during IP insertion, a warning message is issued but the modified modules are written out.

- **-NO_SYNopsys_pragma**

An optional switch that specifies to *not* write out pragmas when the tool writes out the RTL description of EDT logic in either Verilog or VHDL. By default, when synthesis targets Synopsys Design Compiler (-SYnthesis_script DC_Tcl | DC_Shell), a pragma is written

indicating to the synthesis tool that it is a reset signal and should not be merged with other signals.

- **-NO_WRITE_Design**

An optional switch that directs the tool to not write out an EDT-inserted netlist when this command is executed. In insertion mode, you can edit the netlist further and save the design. You must save the design using the [write_design](#) command, and you must name the file using the naming convention *<filename_prefix>_ edt_top_rtl.v* which is referenced by the synthesis script.

- **-Ijtag Off | Data_ports**

An optional switch and value pair that directs the tool to use IJTAG in the generated dofiles to describe the static configuration inputs of the TestKompress IP. Use this option if you are using a dofile to pass IP information from the IP creation phase to the pattern generation phase. These static configuration inputs set, enable, or disable certain features of the TestKompress IP: edt bypass, single chain bypass, low power and edt configuration.

Note

 If you are using the Tessent Core Description (TCD) EDT flow, then you do not need to specify this option. See “[IJTAG and the EDT IP TCD Flow](#)” in the *Tessent TestKompress User’s Manual* for more information. By default, the tool performs the IJTAG mapping automatically and writes out the ICL and PDL files.

For the TCD EDT flow, you can disable the IJTAG mapping using the following command:

```
set_procedure_retargeting_options -ijtag off
```

For details on how to use the IJTAG files for TestKompress ATPG, see section “Setting up EDT IP for IJTAG Integration” in the “[IJTAG Features of ATPG in Tessent Shell](#)” section of the *Tessent IJTAG User’s Manual*.

- **-TSdb [-DESIGN_Id *design_id*]**

An optional and optional switch and string pair that specifies to write the EDT files into the [instruments](#) directory of the [Tessent Shell Data Base \(TSDB\)](#). The name of the TSDB directory defaults to *tsdb_outdir* and is specified using the [set_tsdb_output_directory](#) command.

The optional **-DESIGN_Id *design_id*** switch and string pair specifies the *design_id* value when creating the TSDB directory. Refer to “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763 for complete information. The default *design_id* is that specified by the [set_context](#) command, or the default of that command if “[set_context -design_identifier](#)” is not used. In general, it is not recommended to specify **-DESIGN_id** as part of [write_edt_files](#) since this option overrides the value used by other TSDB-related commands used in that context.

Exporting your instruments using this method during post-scan insertion also enables the [run_synthesis](#) command to synthesize the generated IP.

The EDT Tesson Core Descriptions are also written out. See “[Tesson Core Description \(TCD\)](#)” in the *Tesson TestKompress User’s Manual* for more information. EDT TCDs created using this switch will be automatically loaded into the tool and do not need to be explicitly read.

This option is mutually exclusive with the use of *filename_prefix*. See [Example 12](#) for a usage example.

Examples

Example 1

The following external flow example writes the files for EDT logic including bypass circuitry, and uses the filename prefix, *my_design* to name the output files:

```
write_edt_files my_design -verilog -replace -nortl_prefix

// Writing my_design_edt.v
// Writing my_design_edt_top.v
// Writing my_design_core_blackbox.v
// Writing my_design_dc_script.scr
// Writing my_design_edt.dofile
// Writing my_design_edt.testproc
// Writing my_design_bypass.dofile
// Writing my_design_bypass.testproc
// Writing my_design_edt.icl
// Writing my_design_edt.pdl
// Writing my_design_edt.tcd
```

The module and instance names in the RTL generated by the preceding example all begin with the *edt_* prefix.

Example 2

The following internal flow example shows the files written when the EDT logic is placed inside the core netlist including a reduced version of the input netlist for synthesis. The files resulting from the *-Reduced_netlist* switch are in bold.

```
write_edt_files my_design -verilog -replace -nortl_prefix -reduced_netlist -insertion dc

// Writing my_design_edt.v
// Writing my_design_dc_script.scr
// Writing my_design_reduced_netlist.v
// Writing my_design_rest_of_netlist.v
// Writing my_design_edt.dofile
// Writing my_design_edt.testproc
// Writing my_design_bypass.dofile
// Writing my_design_bypass.testproc
// Writing my_design_edt.icl
// Writing my_design_edt.pdl
// Writing my_design_edt.tcd

write_design -output_file my_design_new.v
```

Example 3

Assume the output file resulting from synthesis of *my_design_reduced_netlist.v* is named *my_design_edt_top_gate.v*. The following example shows how to invoke the tool, set the context for pattern generation, and load the Verilog files:

```
tessent -shell <other arguments>
SETUP> set_context patterns -scan
SETUP> read_verilog my_design_edt_top_gate.v my_design_rest_of_netlist.v
SETUP> read_cell_library tessent_cell.tcellib
SETUP> set_current_design
```

Example 4

The following internal flow example writes out EDT logic in multiple files, one Verilog module per file. The EDT logic is placed internal to the design core, and the top-module name in the design is *core1*:

```
write_edt_files results/created -single_module_per_file -verilog

// Writing results/core1_edt_decompressor.v
// Writing results/core1_edt_spatial_compactor_8.v
// Writing results/core1_edt_controlled_decoder_3_to_8.v
// Writing results/core1_edt_compactor.v
// Writing results/core1_edt_bypass_logic.v
// Writing results/core1_edt.v
// Writing results/core1_edt_mux_2_to_1.v
// Writing results/created_dc_script.scr
// Writing results/created_edt.dofile
// Writing results/created_edt.testproc
// Writing results/created_bypass.dofile
// Writing results/created_bypass.testproc
// Writing results/created_edt.icl
// Writing results/created_edt.pdl
// Writing results/created_edt.tcd

write_design -output_file results/created_new.v
```

Example 5

The following external flow example shows how the *-Rtl_prefix* switch affects the filenames when the EDT logic is placed external to the design core, the EDT logic is written in multiple files, and the design has a top module named *mtm2*:

```
write_edt_files multiple -single_module_per_file -verilog -rtl_prefix designX_blockY
```

```
// Writing designX_blockY_edt_decompressor.v
// Writing designX_blockY_edt_spatial_compactor_8.v
// Writing designX_blockY_edt_controlled_decoder_3_to_8.v
// Writing designX_blockY_edt_compactor.v
// Writing designX_blockY_edt.v
// Writing designX_blockY_edt_pinshare_logic.v
// Writing designX_blockY_edt_top.v
// Writing mtm2_blackbox.v
// Writing multiple_dc_script.scr
// Writing multiple_edt.dofile
// Writing multiple_edt.testproc
// Writing designX_blockY_edt.icl
// Writing designX_blockY_edt.pdl
// Writing designX_blockY_edt.tcd
```

Example 6

The following internal flow example shows the files created when the tool inserts EDT logic into the design core with the *-Insertions* switch:

write_edt_files results/created -verilog -rep -insertions

```
// Writing results/created_edt.v
// Writing results/created_dc_script.scr
// Writing results/created_edt.dofile
// Writing results/created_edt.testproc
// Writing results/created_bypass.dofile
// Writing results/created_bypass.testproc
// Writing results/created_edt.icl
// Writing results/created_edt.pdl
// Writing results/created_edt.tcd
// -----
// Begin EDT logic insertion
// -----
// Inserting EDT logic in the design.
// EDT logic insertion completed.
```

write_design -output_file results/created_new.v

The output file *created_edt_top_rtl.v* contains the EDT logic inserted design without the EDT logic description.

Example 7

The following external flow example writes out timing files in addition to the default EDT files and specifies *created* for the naming prefix:

write_edt_files created -timing_constraints

write_edt_files

```
// Writing created_edt.v
// Writing created_edt_top.v
// Writing created_core_blackbox.v
// Writing created_dc_script.scr
// Writing created_edt.dofile
// Writing created_edt.testproc
// Writing created_bypass.dofile
// Writing created_bypass.testproc
// Writing created_edt_shift_sdc.tcl
// Writing created_bypass_shift_sdc.tcl
// Writing created_edt_slow_capture_sdc.tcl
// Writing created_edt.icl
// Writing created_edt.pdl
// Writing created_edt.tcd
```

Example 8

The following internal flow example shows the files created when the tool inserts EDT logic into the design core and the *-synthesis_script* switch is specified with the *rtl_compiler* option. The file resulting from the *-synthesis_script rtl_compiler* switch and option is in **bold**.

```
write_edt_files created -verilog -synthesis_script rtl_compiler -replace

// Writing created_edt.v
// Writing created_rtlc_script.scr
// Writing created_edt.dofile
// Writing created_edt.testproc
// Writing created_bypass.dofile
// Writing created_bypass.testproc
// Writing created_edt.icl
// Writing created_edt.pdl
// Writing created_edt.tcd

// -----
// Begin EDT logic insertion
// -----
// Inserting EDT logic in the design.
// EDT logic insertion completed.
// Note: The synthesis script assumes the modified design will be saved
// in file 'created_edt_top_rtl.v'.
// Update the synthesis script when saving to a different file.
```

write_design -output_file created_new.v

Example 9

In this example, assume the top level of the design has the three modules *cpu_core*, *clkblk*, and *padring* instantiated. The following commands write out the top-level module of the design but exclude the *cpu_core*, *clkblk*, and *padring* modules. After executing these commands, the output netlist contains only the top module description with the instances of these three modules:

```
write_edt_files created -replace
write_design -output_file created_edt_top_rtl.v -exclude_modules {cpu_core clkblk padring}
```

Assume the input design for this example is contained in two files: *top.v*, which contains only the top-level module, and *chip.v*, which contains the other three modules. The following

example invokes the tool, sets the context for IP creation, loads the Verilog files, and runs the dofile, which is assumed to include the above write_edt_files and write_design commands:

```
tessent -shell
SETUP> set_context dft -edt
SETUP> read_verilog top.v chip.v
SETUP> read_cell_library tessent_cell.tcellib
SETUP> set_current_design
SETUP> dofile ip_creation.do
```

After performing logic synthesis of the EDT logic, the output file is *created_edt_top_gate.v*. The command-line invocation for pattern generation uses the synthesized EDT logic file *created_edt_top_gate.v* as well as the file *chip.v* that contains the description of the modules excluded during IP creation.

```
tessent -shell
SETUP> set_context patterns -scan
SETUP> read_verilog created_edt_top_gate.v chip.v
SETUP> read_cell_library tessent_cell.tcellib
SETUP> set_current_design
SETUP> dofile pattern_gen.do
```

Example 10

In this example, the write_edt_file command writes out RTL with synopsys pragma:

write_edt_files created -replace

In Verilog, the pragma is included as a comment near the *always* block that describes the target registers as shown here:

```
*****  
** Module: m8051_edt_decompressor  
** Tessent TestKompress version: v9.1  
** IP version: 5  
** Date: Thu Aug 19 14:58:39 2010  
***** /
```

```

module m8051_edt_decompressor (edt_clock, edt_update, edt_channels_in,
    edt_scan_in);
    input edt_clock;
    input edt_update;
    input [ 1:0] edt_channels_in;
    output [15:0] edt_scan_in;
    reg      [15:0] edt_scan_in;
    reg      [11:0] lfsm_vec;
    reg      [11:0] lfsm_vec_lockup;
    // synopsys sync_set_reset edt_update
    always @(posedge edt_clock)
    begin : lfsm
        if (edt_update == 1'b1) begin
            lfsm_vec <= 12'b00000000000000;
        end
        else begin
            lfsm_vec[ 0] <= lfsm_vec[ 1];
            lfsm_vec[ 1] <= lfsm_vec[ 2] ^ edt_channels_in[1];
            lfsm_vec[ 2] <= lfsm_vec[ 3];
            lfsm_vec[ 3] <= lfsm_vec[ 4] ^ edt_channels_in[1];
            lfsm_vec[ 4] <= lfsm_vec[ 5];
            lfsm_vec[ 5] <= lfsm_vec[ 6] ^ edt_channels_in[1];
            lfsm_vec[ 6] <= lfsm_vec[ 7] ^ lfsm_vec[ 3];
            lfsm_vec[ 7] <= lfsm_vec[ 8] ^ edt_channels_in[0];
            lfsm_vec[ 8] <= lfsm_vec[ 9] ^ lfsm_vec[ 4];
            lfsm_vec[ 9] <= lfsm_vec[10] ^ edt_channels_in[0];
            lfsm_vec[10] <= lfsm_vec[11] ^ lfsm_vec[ 1];
            lfsm_vec[11] <= lfsm_vec[ 0] ^ edt_channels_in[0];
        end
    end
    ...

```

In VHDL, the pragma is specified as an attribute from the Synopsys library in the declarations section of the architecture description as shown here:

```
-----  
-- Module: m8051_edt_decompressor  
-- Tesson TestKompress version: v9.1  
-- IP version: 5  
-- Date: Thu Aug 19 15:01:15 2010  
-----  
LIBRARY ieee;  
LIBRARY synopsys;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
USE synopsys.attributes.all;  
USE work.all;  
ENTITY m8051_edt_decompressor IS  
  PORT (  
    edt_clock      : IN      std_logic;  
    edt_update     : IN      std_logic;  
    edt_channels_in : IN      std_logic_vector(1 DOWNTO 0);  
    edt_scan_in    : OUT     std_logic_vector(15 DOWNTO 0)  
  );  
END m8051_edt_decompressor;  
ARCHITECTURE rtl OF m8051_edt_decompressor IS  
  SIGNAL lfsm_vec      : std_logic_vector(11 DOWNTO 0);  
  SIGNAL lfsm_vec_lockup : std_logic_vector(11 DOWNTO 0);  
  ATTRIBUTE sync_set_reset of edt_update : SIGNAL is "true";  
BEGIN  
  lfsm: PROCESS (edt_clock)  
  BEGIN  
    IF (edt_clock = '1') AND (edt_clock'EVENT) THEN  
      IF (edt_update = '1') THEN  
        lfsm_vec <= (OTHERS => '0');  
      ELSE  
        lfsm_vec( 0 ) <= lfsm_vec( 1 );  
        lfsm_vec( 1 ) <= lfsm_vec( 2 ) XOR edt_channels_in(1);  
        lfsm_vec( 2 ) <= lfsm_vec( 3 );  
        lfsm_vec( 3 ) <= lfsm_vec( 4 ) XOR edt_channels_in(1);  
        lfsm_vec( 4 ) <= lfsm_vec( 5 );  
        lfsm_vec( 5 ) <= lfsm_vec( 6 ) XOR edt_channels_in(1);  
        lfsm_vec( 6 ) <= lfsm_vec( 7 ) XOR lfsm_vec( 3 );  
        lfsm_vec( 7 ) <= lfsm_vec( 8 ) XOR edt_channels_in(0);  
        lfsm_vec( 8 ) <= lfsm_vec( 9 ) XOR lfsm_vec( 4 );  
        lfsm_vec( 9 ) <= lfsm_vec(10) XOR edt_channels_in(0);  
        lfsm_vec(10) <= lfsm_vec(11) XOR lfsm_vec( 1 );  
        lfsm_vec(11) <= lfsm_vec( 0 ) XOR edt_channels_in(0);  
      END IF;  
    END IF;  
  END PROCESS lfsm;  
  ...  
  
```

Example 11

The following example shows two commands that do not include the synopsys pragma:

`write_edt_files noprag1 -synthesis_script rtl_compiler`

write_edt_files noprag2 -no_synopsys_pragma

Example 12

The following example uses the -tsdb option and default value of the [set_tsdb_output_directory](#) command. The module name prefix is “*current_design_design_id_tessent_*”. See the [set_context](#) -design_identifier command for more information about the design identifier. See the [instruments](#) directory section for details about the instrument container directory.

write_edt_files -tsdb -replace

```
// Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.v
// Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_edt.synthesis_dictionary
// Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.icl
// Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.pdl
// Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.tcd
// -----
// Begin EDT logic insertion
// -----
// Inserting EDT logic in the design.
// EDT logic insertion completed.
// Writing modified netlist ./tsdb_outdir/dft_inserted_designs/
m8051_gate.dft_inserted_design/m8051.vg_no_instruments.
// Writing interface netlist ./tsdb_outdir/dft_inserted_designs/
m8051_gate.dft_inserted_design/m8051.v_interface
```

Related Topics

[report_edt_configurations](#)

[report_edt_pins](#)

[write_design](#)

[set_edt_options](#)

[set_edt_pins](#)

write_failing_paths

Context: patterns -scan_diagnosis

Mode: analysis

Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.

Usage

```
write_failing_paths [-DIAGnosis_report diagnosis_report_file_name]  
[-Pattern number] [-ENDpoints [Cell_Ins_Name1 PO_Pin_Name1 ...]]  
[-OUTput name [-REPlace]]
```

Description

Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.

You must specify performing at-speed diagnosis using the “set_diagnosis_options -at_speed on” command before using the write_failing_paths command. The write_failing_paths command issues a warning message if at-speed diagnosis is off. If you use the following commands:

```
read_patterns pat.file  
set_diagnosis_options -at_speed on  
write_failing_paths -diagnosis_report report.diag
```

Then you don't have to use the diagnose_failures command at all.

Arguments

- **-DIAGnosis_report *diagnosis_report_file_name***
An optional switch and string pair specifying the diagnosis report filename from which failure diagnosis is processed.
- **-Pattern *number***
An optional switch and integer pair specifying the failing pattern number for which failing paths are reported.
- **-ENDpoints *Cell_Ins_Name1 PO_Pin_Name1 ...***
An optional switch and report failing paths that end at the given cell instance names or PO pin names. The end cell instances are the state elements where at-speed failures are captured. Default: report failing paths for all cells or POs.
- **-OUTput *name***
An optional switch and string pair specifying the filename (*name*) where failing paths are reported. The tool will report failing paths to the screen and log file.

Examples

Example Failing Path Report

```
// Symptom = 1
// Scan pattern: 42
// Suspect = 1
PATH "path1" =
PIN /CORE/TP004/P1001/PC00W33_0/Q;
PIN /CORE/TM00T/AL011/U117/B1 ;
PIN /CORE/TM00T/AL011/U117/OEQ ;
PIN /CORE/TM00T/AL011/U39/I1 ;
PIN /CORE/TM00T/AL011/U39/O ;
PIN /CORE/TM00T/AL011/AP00A00_0/D ;
END ;
— skipping paths "path2-4" in this example —
PATH "path5" =
PIN /CORE/TP004/P1001/PC00W55_0/Q ;
PIN /CORE/TM00T/AL011/U116/B2 ;
PIN /CORE/TM00T/AL011/U116/OEQ ;
PIN /CORE/TM00T/AL011/U54/B ;
PIN /CORE/TM00T/AL011/U54/OB ;
PIN /CORE/TM00T/AL011/AP00A00_0/D ;
END ;
// Suspect = 2
// "path5"
// Scan pattern: 121
// Suspect = 1
// "path1"
// "path2"
// Suspect = 2
// "path5"
```

The above example shows 5 failing paths for *symptom 1, suspect 1* at scan pattern 42. Detailed description of the PATH definition can be found in the FastScan manual. Also shown is that for same scan pattern, 42, there is only one failing path for *symptom 1, suspect 2*, which is equivalent to *path5* that was defined earlier. For scan pattern 121, there are two patterns for *suspect 1 of symptom 1*, and they are path1 and path2 which are defined earlier; there is one path, path5 for *suspect 2 of symptom 1*.

Example 1

This example reports warning message because this command can only used for at-speed diagnosis.

```
set_diagnosis_options -at_speed off
diagnose_failures chip1.fail
write_failing_paths
```

Example 2

This example reports the at-speed failing paths for the diagnosis result of *chip1.fail* on the screen.

```
set_diagnosis_options -at_speed on
```

diagnose_failures chip1.fail

write_failing_paths

Example 3

In this example at-speed failing paths of scan pattern 5 for *chip1.fail* and the failing paths of scan pattern 10 for *chip2.fail* are printed on the screen.

set_diagnosis_options -at_speed on

diagnose_failures chip1.fail -output chip1.diag

write_failing_paths -pattern 5

diagnose_failures chip2.fail -output chip2.diag

write_failing_paths -pattern 10

Example 4

The at-speed failing paths of diagnosis *chip1.diag* are printed on the screen. The *-end_cell* switch reports only failing paths ending at cell “/cpu/exu/pmi/ix926,” where “/cpu/exu/pmi/ix926” is the name of the cell where at-speed failures are captured.

set_diagnosis_options -at_speed on

diagnose_failures chip1.fail -output results/chip1.diag

write_failing_paths -end_cell /cpu/exu/pmi/ix926

Example 5

The critical paths of diagnosis results for *chip1.fail* are written to file named *chip1.path*.

set_diagnosis_options -at_speed on

diagnose_failures chip1.fail -output results/chip1.diag

write_failing_paths -output chip1.path

Related Topics

[diagnose_failures](#)

[set_design_sources](#)

write_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis,
patterns -failure_mapping

Mode: analysis

Injects faults and/or writes failing pattern results to a file.

Usage

```
write_failures failure_filename [-Replace]
[[{pin.pathname -Stuck_at {0 | 1}}...] |
[pin.pathname {-SLOW_TO_RISE | -SLOW_TO_FALL}] ]
[-Pdet | -Exact] [-Max max_failed_scan_pat_num] [-CYCle]
[-Faulty_chain faulty_chain_name faulty_chain_type {-Cell faulty_cell_id...}...]
```

Description

Injects faults and/or writes failing pattern results to a file.

The resulting file can be used to perform a diagnosis of the failures with Tesson Diagnosis

By default, the write_failures command performs a good simulation of the current design and test pattern source and writes the failing pattern information to the display. write_failures also does not produce the cell paths by default.

You can also use this command to inject faults into the design, simulate them, and write the resultant failures into the failure file for diagnosis. When faults are injected, a fault simulation is run with those values. The fault is injected with respect to the scan cell that is closest to the scan output.

For failure mapping, the tool creates core-level failure files by reverse mapping the top-level failures that were obtained by applying the top-level patterns generated with scan pattern retargeting. After reading a valid top-level failure file, the command extracts core-level failures for all failing cores and saves them into separate files that are named based on *failure_filename* and the naming convention as described below. The command strips the Verilog escaped characters too.

The tool names the core-level failure file for a failing core by appending its adjusted core instance name, its core module name, and its core test mode to the given failure file basename. A triple underscore ("___") is used to separate the core failure file basename from the other appended names, which are separated by double underscores ("__"). To convert the hierarchical instance name to a valid file name, the tool replaces "/" characters in the core instance name with a period character (".").

For example, the tool would save the core-level failure file for core instance core_10 of core module MyCore, tested in internal mode, to the name *./core_flogs/fail1.top___core_10__MyCore__internal*. Likewise, the tool would save the core-level failure

file for the core instance A/B/core1 of core module “MyCore, tested in internal mode, to the name *./core_flogs/fail1.top__A.B.core1__MyCore_internal*.

Arguments

- ***failure_filename***
A required string that specifies the name of a file to write the failing pattern information to.
- **-Replace**
An optional switch that directs the tool to overwrite an existing *failure_file*. Use this if the specified *failure_file* already exists, and you want to update the contents.
- ***pin.pathname* -Stuck_at {0 | 1}**
An optional, repeatable string, switch, and literal triplet that specifies the location and the value of a logic failure to inject into the design as follows:
 - pin.pathname* — A string that specifies the pin pathname of the location to inject the fault.
 - Stuck_at 0 | 1 — A switch and literal that specifies the stuck-at value for the fault. The stuck-at options include:
 - 0 — *stuck-at-0* fault
 - 1 — *stuck-at-1* fault
- ***pin.pathname* {-SLOW_TO_RISE | -SLOW_TO_FALL}**
An optional repeatable string and switch pair that injects single transition fault to support at-speed diagnosis using the following criteria:
 - pin.pathname* — A string that specifies the pin pathname of the location to inject the transition fault.
 - SLOW_TO_RISE — A literal that specifies failure at the 0->1 transition.
 - SLOW_TO_FALL — A literal that specifies failure at the 1->0 transition.
 This argument supports injecting a single transition fault. Additionally, when the transition fault is injected, other faults (for example, stuck-at logic faults or chain faults) cannot be injected together, specifically this argument does not support a mixture of a transition fault with other faults.
- **-Pdet**
An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected for a specified fault. By default, only the binary detections are output to the file.
- **-Exact**
An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected or a binary value where an X value is expected for a specified fault. By default, only the binary detections are output to the file.

- **-Max** *max_failed_scan_pat_num*

An optional switch and integer pair that specifies a maximum number of failing patterns to write for the specified faults. When the maximum number is reached, the simulation stops. By default, all failing patterns are written.

- **-CYCle**

An optional switch that writes the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the “[Input File Requirements](#)” chapter in the *Tessent Diagnosis User’s Manual*.

- **-Faulty_chain** *faulty_chain_name* *faulty_chain_type*

An optional repeatable switch and argument pair that specifies the names and fault types of the chain faults to inject into a design and simulate. Argument options include:

faulty_chain_name — String that specifies the name of the faulty chain.

fault_type — literal that specifies the fault type associated with the chain. Argument options include:

STUCK_AT_1

STUCK_AT_0

SLOW_TO_FALL

SLOW_TO_RISE

FAST_TO_RISE

FAST_TO_FALL

FAST

- **-Cell** *faulty_cell_id*

An optional switch and repeatable integer that specifies the faulty cells within the specified faulty chain.

Examples

Example 1

The following example switches to analysis system mode, loads a test pattern source, injects a stuck-at 1 failure at the i_1006/o logic in the specified a pattern-based failure file.

```
set_system_mode analysis
read_patterns file1
write_failures fail1 i_1006/o -stuck_at 1
// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec
```

The output file looks like:

```

4 chain1 3 L H //uINTR/ix476
6 /D_OUT(0) L H //uINTR/ix446
7 chain1 3 L H //uINTR/ix356
...
29 /D_OUT(1) L H //uINTR/ix358
29 chain1 3 L H //uINTR/ix362
31 /D_OUT(1) L H //uINTR/ix601
31 /D_OUT(2) L H //uINTR/ix602
last_pattern_tested 35

```

Example 2

The following example shows a pattern-based output file written from Tesson TestKompress (4X compression).

```

0 edt_channel1 2 H L // chain1---2---DFF_reg0
// chain2---2---DFF_reg1
// chain3---2---DFF_reg2
// chain4---2---DFF_reg3
1 edt_channel1 2 L H // chain1---2---DFF_reg0
2 output[1] H L // PO
.....

```

Example 3

The following example shows the cycle-based output file written from Tesson TestKompress (4X compression).

```

20 ch_out1 H L //pat 0 channel1 2 chain1---2---DFF_reg0
// chain2---2---DFF_reg1
// chain3---2---DFF_reg2
// chain4---2---DFF_reg3
.....

```

Example 4

The following example reverse maps a top-level failure file into core-level failure files and saves them to the `./core_flogs` directory based on the naming convention described above.

```

write_failures ./core_flogs/top_flog1 -rep
// command: write_failures ./core_flogs/top_flog1 -rep
// Note: Core level faillog
//      (./core_flogs/top_flog1__logic_die_BOT_inst_logic_die_internal)
//      is created for core ( logic_die_BOT_inst_logic_die_internal ).
// Note: Extract 1 core faillog for 1 failing core from top level
//      faillog ( top_flogs/top_flog1 ).
```

Related Topics

[read_failures](#)
[read_patterns](#)

[**report_failures**](#)

[**diagnose_failures**](#)

write_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Writes path definitions or bridge entries from the internal list to a file.

Usage

Path Delay Faults Usage:

```
write_fault_sites filename [-Replace] [-All | {-Path gate_id_begin gate_id_end}]
```

Bridge Faults Usage:

```
write_fault_sites filename [-Replace] [-All | net.pathname_pair / bridge_name... |  
 net.pathname... | instance_name...] [-NAME | -SINGLE]  
 [-NET.pathname | -PIN.pathname] [-NOEQ]
```

Description

Writes path definitions or bridge entries from the internal list to a file.

The write_fault_sites command writes the specified fault sites from the internal list to a file. Depending on the active fault model, fault sites may be defined with path definitions or bridge entries. Fault sites are loaded into the internal list with the read_fault_sites command.

For more information on bridge faults and the bridge fault definition file, refer to “[The Static Bridge Fault Model](#)” in the *Tessent Scan and ATPG User’s Manual*.

For more information on path delay faults and the path definition file, refer to “[Path Delay Test Set Creation](#)” in the *Tessent Scan and ATPG User’s Manual*.

By default, this command writes out the loaded fault attributes.

Path Delay Specifics

The write_fault_sites command also writes unambiguous paths that you previously added to the internal path list using the [add_ambiguous_paths](#) command. When writing derived unambiguous paths, write_fault_sites cannot list a gate if it does not have a pin pathname associated with it. For some designs, gates without associated pin pathnames could be fairly common. Therefore, refer to the written unambiguous paths for launch and capture points, but be aware the listing may not show all the gates in between.

Note

 If you get ambiguous path warnings, then add_ambiguous_paths, those paths when written out lose the information about which of the ambiguous paths they were. Consequently, you cannot reload them and get back to the same set of paths.

Arguments

- ***filename***
A required string that specifies the name of the file to which the tool should write the path definition or bridge entry information.
- **-Replace**
An optional switch that replaces the contents of the file, *filename*, if it already exists.
- **-All**
An optional switch that displays all fault definitions for the currently loaded path definitions or bridge entries. This is the default.
- **-Path *gate_id_begin* *gate_id_end***
An optional switch and two-integer triplet that specifies a particular path or portion of a path whose definition you want to write. Use this argument to write_paths that were not defined in a path definition file and, therefore, were not loaded using the `read_fault_sites` command.
The two integers specify two gate identification numbers that indicate the beginning and end of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.
The value of *gate_id_begin* or *gate_id_end* is the unique gate identification number that the tool automatically assigns to every gate within the design during the model flattening process.
- ***net_pathname_pair***
A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.
- ***bridge_name***
A repeatable string that specifies the name of a specific bridge entry.
- ***net_pathname***
A repeatable string that specifies a single net or pin pathname contained in a bridge entry.
- ***instance_name***
A repeatable string that specifies an instance pathname contained in a bridge entry.
- **-NAME**
An optional switch that specifies all entered strings are bridge names.
- **-SINGLe**
An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.
- **-NET_pathname**
An optional switch that specifies to write net pathnames for bridge entries. This is the default.

- **-PIN.pathname**
 An optional switch that specifies to write pin names for bridge entries instead of net pathnames.
- **-NOEQ**
 An optional switch that writes only the representative fault information: net pair and fault categories.

Examples

Example 1

The following path delay example writes to the file my_path_definitions using the path definition file format, the pins in the specified path:

write_fault_sites /user/design/my_path_definitions -path 180 178

The following shows an example of the contents of a path definition file:

```
PATH "path0" =
PIN /I$6/Q + ;
PIN /I$35/B0 + ;
PIN /I$35/C0 + ;
PIN /I$1/I$650/IN + ;
PIN /I$1/I$650/OUT - ;
PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge example reports all bridge entries in the current fault site list, then writes one entry (including its attributes) to the file my_bridge_definitions using the bridge fault definition file format:

```
report_fault_sites

BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U0.BRIDGE";
}
```

write_fault_sites /user/design/my_bridge_definitions U5.BRIDGE -name

The following shows the contents of the resultant *my_bridge_definitions* file:

```
VERSION      1.1

FAULT_TYPE="BRIDGE_STATIC_4WAY_DOM"

BRIDGE {
    NET1  = "/G4";
    NET2  = "/G10";
    FAULTS = {DS, DS, DS, DS};
    WEIGHT = 1.0000;
    DISTANCE = 4.96575000e-06 um;
    PARALLEL_run = 7.30000000e-05 um;
    X_COORDINATE = 174500;
    Y_COORDINATE = 570000;
    LAYER = "1";
    NAME = "U5.BRIDGE";
}
```

Related Topics

[add_ambiguous_paths](#)
[add_faults](#)
[delete_fault_sites](#)
[read_fault_sites](#)
[report_fault_sites](#)

write_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Writes fault information from the current fault list to a file.

Usage

Path Delay Usage: write_faults ***file_name*** [-Replace] [-Both | -Rise | -Fall]
 [-Class *fault_class*[*fault_subclass*]] [*object_pathname*] [-NO_Subclass]
 [-CHANGE_classification_to *fault_class*[*.subclass*]]
 [-VERilog | -VHdl] [-FORmat {Basic | Mtfi}]

Bridge Usage: write_faults ***file_name*** [-Replace] [-NO_Subclass] [-FORmat {Basic | Mtfi}]
 {[*net_pathname_pair* | *bridge_name* ... | *net_pathname* ... | *instance_name* ...]
 [-Class *fault_class*[*fault_subclass*]]
 [-NAME | -SINGLE] [-NET *pathname*] [-PIN *pathname*] [-NOEQ]
 [-VERilog | -VHdl] [-CLOCK_Domains {ALL | *clock_pathname* ...}]
 [-NO_EQUIvalent_clocks] [-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
 [-CAPture_procedures {ALL | *capture_procedure_name* ...}]} |
 {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}

Toggle/Iddq Usage: write_faults ***file_name*** [-Replace] [-NO_Subclass]
 [-FORmat {Basic | Mtfi}] [-FRom *pin*...] [-TThrough *pin*...] [-TO *pin*...]
 [-STOP_at {sequential_elements | *scan_cells* | *ports_only*}]
 {[[-Class *fault_class*[*fault_subclass*...] [-NOEQ] [-Stuck_at [01 | 0 | 1]]]
 [-CHANGE_classification_to *fault_class*[*.subclass*]]
 [*object_pathname* ...] [-Hierarchy *integer*] [-Min_count *integer*]}
 [-VERilog | -VHdl] [-CELL_name]
 [-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUIvalent_clocks]]
 [-CAPture_procedures {ALL | *capture_procedure_name* ...}]} |
 {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]} |
 {-UNlisted}

Stuck/Transition Usage: write_faults ***file_name*** [-Replace] [-NO_Subclass]
 [-FORmat {Basic | Mtfi}] [-FRom *pin*...] [-TThrough *pin*...] [-TO *pin*...]
 [-STOP_at {sequential_elements | *scan_cells* | *ports_only*}]
 {[[-Class *fault_class*[*fault_subclass*...] [-NOEQ] [-Stuck_at [01 | 0 | 1]]]
 [-CHANGE_classification_to *fault_class*[*.subclass*]]
 [*pin_pathname* ...] [-Hierarchy *integer*] [-Min_count *integer*]}
 [-VERilog | -VHdl] [-CELL_name]
 [-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUIvalent_clocks]]
 [-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
 [-CAPture_procedures {ALL | *capture_procedure_name* ...}] [-DElay_data]
 [-TIMING_CRITICAL]} | {[-SCAN_Enable] [-CLOCK_Cones] [-IO]
 [-ASYnchronous_controls]} | {-UNlisted}

UDFM Usage: write_faults *file_name* [-Replace] [-NO_Subclass] [-FORmat Mtfi]
 { -Class *fault_class*[.*fault_subclass*]... | *instance.pathname* | [-Hierarchy *integer*]
 [-CHANGE_classification_to *fault_class*[.*subclass*]]
 [-NOEQ] [-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUIvalent_clocks]
 [-EXCLUDEFAULTS_BETWEEN_SYNC_clock_domains]] {[[-Udfm_type *name*]
 [-INstance *name*] [-MModule *name*]}} | {-UNlisted}
 [-FRom pin...] [-THrough pin...] [-TO pin...]
 [-STOP_at {sequential_elements | scan_cells | ports_only}]}

Description

Writes fault information from the current fault list to a file.

For more information about the MTFI file format, refer to the [Tessent Scan and ATPG User's Manual](#).

The write_faults command is similar to the report_faults command, except that the data is written into a file. You can review or modify the file and later load the information into the fault list with the read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any of the optional arguments, write_faults writes information on all the known faults to the file.

The file contains the following columns of information for each fault:

- fault value—The fault value may be either 0 (for stuck-at-0 or slow-to-rise transition faults) or 1 (for stuck-at-1 or slow-to-fall transition faults).
- fault code—A code name that indicates the lowest-level fault class assigned to the fault.
- fault site—The pin pathname of the fault site.
- cell name (optional)—The name of the cell corresponding to the fault (included only if you used the -Cell_name argument).

You can use the -Hierarchy option to write a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option which specifies the minimum number of faults that must be in a hierarchical level before writing.

You can choose to display either collapsed or uncollapsed faults by using the set_fault_mode command.

Note that this command can write specific AU.TC and AU.PC faults when you previously issue the “[report_statistics](#) -detailed_analysis” command. If you want to report specific AU.TC faults, you must specify gate IDs instead of the full pin pathname. If you want to report specific AU.PC

faults, you must specify the full pin pathname. See [Example 5](#) for an example of writing specific AU.TC and AU.PC faults.

Arguments

- ***file_name***

A required string that specifies the name of the file where the tool is to write the fault information. The *file_name* must be the first argument.

- **-Replace**

An optional switch that replaces the contents of *file_name* if the file already exists.

- **-Class *fault_class*[.*fault_subclass*]**

An optional switch and literal pair that specifies the fault class or fault class/sub-class to write. You should use the code when specifying a fault class or sub-class. If you do not use this switch, the default is to write all fault classes.

This command can write specific AU.TC and AU.PC faults when you previously issue the “[report_statistics -detailed_analysis](#)” command. For AU.TC faults you must specify gate IDs rather than the full pin pathname. For AU.PC faults, you must specify the full pin pathname. For more information, refer to “[Example 5](#)” on page 2578.

- **-change_classification_to *fault_class*[.*subclass*]**

By default, the tool writes the fault class (and subclass) of the faults learned by the tools (for example, AU.TI or DS). This option allows you to select the faults and force these faults to be written as the specified class (and subclass). For a list of fault classes and subclasses refer to “[Fault Sub-classes](#)” in the *Tessent Scan and ATPG User’s Manual*.

- **-FORmat [Basic | Mtfi]**

An optional switch and literal pair that specifies the format of the fault file. By default, the command writes out in MTFI format. For more information about MTFI, refer to the “[MTFI File Format](#)” chapter in the *Tessent Scan and ATPG User’s Manual*.

- **-FRom pin...**

This option sets the start pin(s) of the cone of logic used to select faults to write. When specified, the tool performs forward cone tracing from the specified pin(s) until it encounters a stop point. You can specify one or more instance names, rather than pin names. When you specify an instance name with -from, its output pins are used when defining the cone.

- **-TThrough pin...**

This option sets the through pin(s) of the cone of logic used to select faults to write. When specified, the tool performs forward and backward cone tracing from the specified pin(s) until it encounters a stop point in both directions. You can specify one or more instance names, rather than pin names. When you specify an instance name with -through, its output pins are used when defining the cone.

- **-TO pin...**

This option sets the end pin(s) of the cone of logic used to select faults to write. When specified, the tool performs backward cone tracing from the specified pin(s) until it encounters a stop point. You can specify one or more instance names, rather than pin names. When you specify an instance name with **-to**, its input pins are used when defining the cone.

Note

 When **-from**, **-to**, and **-through** are used with other switches, the selected faults will be the intersection of the cones traced for all specified switches.

- **-STOP_at {sequential_elements | scan_cells | ports_only}**

This option sets the stop condition for tracing of the cone of logic used to select faults to write. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify “**scan_cells**”, the cone includes non-scan cells and tracing stops only at scan cells. When you specify “**ports_only**”, the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

- **-UNlisted**

Writes only the unlisted fault data. If there are multiple grayboxed instances in the design, this switch writes accumulated fault numbers.

- **-NO_Subclass**

Prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when available.

- **-Stuck_at 01 | 0 | 1**

An optional switch and literal pair that specifies the stuck-at or transition faults you want to write. The choices are as follows:

01 — A literal that specifies for the tool to write both stuck-at-0 and stuck-at-1 faults for stuck-at faults; or to write both slow-to-rise and slow-to-fall faults for transition faults. This is the default.

0 — A literal that specifies to write only the stuck-at-0 faults (slow-to-rise faults for transition faults).

1 — A literal that specifies to write only the stuck-at-1 faults (slow-to-fall faults for transition faults).

- **object_pathname**

An optional repeatable string that specifies the list of pins, instances, or delay paths whose faults you want to write.

- **-Hierarchy integer**

An optional switch and integer pair that specifies the maximum fault class hierarchy level for which you want to write a hierarchical summary of the faults.

- **-Min_count *integer***
 An optional switch and integer pair that you can use with the -Hierarchy option and that specifies the minimum number of faults that must be in a hierarchical level to write a hierarchical summary of the faults. The default is 1.
- **-NOEQ**
 - For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that turns off the display of “EQ” as the fault class for any equivalent faults; the fault class displayed is then that of the representative fault. When you do not specify this switch, the tool displays an “EQ” as the fault class for any equivalent faults. This switch is meaningful only when the [set_fault_mode](#) command is set to Uncollapsed. For more information about representative and equivalent faults, see “[Fault Collapsing](#)” in the *Tessent Scan and ATPG User’s Manual*.
 - For bridge faults only, an optional switch that writes only the representative bridge fault information: the net pair and fault classes of each corresponding dominant net. The tool does not report physical data.
- **-Both | -Rise | -Fall**
 An optional switch that specifies which faults to write for each path already added via the [read_fault_sites](#) command. Use these switches only for path delay faults.
 - Both - An optional switch that specifies to write both the slow-to-rise and slow-to-fall faults. This is the default.
 - Rise - An optional switch that specifies to write only the slow-to-rise faults.
 - Fall - An optional switch that specifies to write only the slow-to-fall faults.
- **-VERilog | -VHdl**
 An optional switch pair that output the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist independent format.
- **-CELL_name**
 An optional switch that specifies to list, for each reported fault, the corresponding circuit element (Tessent Cell library cell, Verilog primitive, primary input, or primary output) as summarized in [Table 6-25](#). Cell names displayed for equivalent faults are based on the cells associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 6-25. Name Conventions Used by `write_faults -Cell_name`

Corresponding Circuit Element	Listed Cell Name
Tessent Cell library cell	Name of the Tessent Cell library cell
Supported Verilog primitive (for a list, see the <i>Tessent Cell Library Manual</i>)	Name of the Verilog primitive
Primary input or output	“primary_input” or “primary_output”

- ***net.pathname_pair***

A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- ***bridge_name***

A repeatable string that specifies the name of a specific bridge entry.

- ***net.pathname***

A repeatable string that specifies a single net or pin pathname contained in a bridge entry.

- ***instance.name***

A repeatable string that specifies an instance pathname contained in a bridge entry.

- **-NAME**

An optional switch that specifies all entered strings are bridge names.

- **-SINGle**

An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

- **-NET.pathname**

An optional switch that reports net pathnames for bridge entries. This is the default.

- **-PIN.pathname**

An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- **-CLOCK.Domains {ALL | clock.pathname...}**

An optional switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to write only faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock. The argument choices are as follows:

ALL — A literal that specifies all the clocks in the design.

clock.pathname — A repeatable string that specifies a particular clock.

Be aware that a fault written using this switch might also be detectable by an unspecified clock.

The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when adding or deleting faults by clock domain.

- **-NO_EQUIvalent_clocks**

An optional switch that prevents the -Clock_domains switch from writing faults in equivalent clock domains. When specified, only faults within clock domains are considered when adding or deleting faults by clock domain.

- **-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains**

The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when deleting faults in the clock domain.

- **-CAPture_procedures {ALL | *capture_procedure_name*...}**

An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to write_faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

ALL — A literal that specifies all enabled named capture procedures.

capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

See the [set_capture_procedures](#) command description for information about enabling or disabling named capture procedures.

- **-DElay_data**

An optional switch that saves the delay faults including the slacks of the faulty site.

- **-TIMING_CRITICAL**

An optional switch that saves the timing critical faults identified by the tool. Before issuing the write_faults command with this switch, you must issue the [set_atpg_timing](#) command with the -TIMING_CRITICAL switch and *timing_margin%* value.

Note

 In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See “[Timing-Aware ATPG](#)” in the *Tessent Scan and ATPG User’s Manual* for complete information.

- **-SCAN_Enable**

An optional switch that specifies to only write faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch writes all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- **-CLOCK_Cones**

An optional switch that deletes all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fan-out of the clock sources. This switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.

- **-IO**

An optional switch that specifies to write faults that are:

- Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
- Only observed by POs

- **-ASYnchronous_controls**

An optional switch that specifies to only write faults that fan out to Set/Reset ports of state elements and RAMs. This switch applies to a subset of the faults the -CLOCK_Cones switch writes.

- **-Udfm_type *name***

An optional switch and string that specifies a name of a UDFM fault type. Use this option to write the faults associated with a specified UDFM fault type to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-Cell *name***

An optional switch and string pair that specifies the name of a library cell. Use this option to write all faults associated with a specified library cell to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-MOdule *name***

An optional switch and string pair that specifies the name of a module. Use this option to write all faults associated with a specified module to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- **-INSTance *name***

An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

Examples

Example 1

The following example performs an ATPG run, then writes all the untestable faults to a file for review:

```
set_fault_type stuck
set_system_mode analysis
create_patterns
write_faults faultlist -class ut -format basic
```

The content of the *faultlist* file will look similar to the following:

```

0      UU    /ip_scan_clk
1      UU    /ip_scan_clk
0      UU    /ipt_scan_tf_en
1      UU    /ipt_scan_tf_en
0      UU    /ff0/qb
1      UU    /ff0/qb
...

```

Example 2

The following example rewrites all the untestable faults to the file, but includes the name of the cell associated with each fault in the written information:

```
write_faults faultlist -class ut -cell_name -format basic -replace
```

The file now contains an additional column with the cell information:

```

0      UU    /ip_scan_clk  primary_input
1      UU    /ip_scan_clk  primary_input
0      UU    /ipt_scan_tf_en  primary_input
1      UU    /ipt_scan_tf_en  primary_input
0      UU    /ff0/qb  dsffd
1      UU    /ff0/qb  dsffd

```

Example 3

The following command writes out a fault file in MTFI format that contains all faults:

```
write_faults fault_file1 -format mtfi
```

This is an example of a small MTFI file that could result from the previous command:

```

FaultInformation {
    version : 1;
    FaultType Stuck {
        FaultList {
            Format : Identifier, Class, Location;
            Instance ("") {
                1, UC, "in1";
                0, DS, "in1";
                0, DS, "i1/IN0";
                1, DS, "i1/OUT";
                1, DS, "i1/IN0";
                1, AU, "i1/IN1";
                0, UC, "i5/Z";
                1, DS, "out";
                1, DI , "i5/Z";
            }
        }
    }
}

```

Example 4

The following example displays the fault data for *ds* class faults associated with library cells that start with *mux* or *an* to the *my_fault_list* file:

```
write_faults my_fault_list -class ds -cell mux* -cell an*

// Command output
FaultInformation {
    version : 1;
    FaultType (UDFM) {
        FaultList {
            Format : Identifier, Class;
            UdfmType ("intra-cell-bridges_1") {
                Cell ("MUX2") {
                    Instance ("/top/i1/i4") {
                        "F2", DS;
                    }
                }
                Cell ("AND3") {
                    Instance ("/top/i1/i456") {
                        "F2", DS;
                        "F4", DS;
                        "F5", DS;
                        "F7", DS;
                    }
                }
            }
        }
    }
}
```

Example 5

The following example shows how to write specific AU.TC and AU.PC faults:

```
report_statistics -detailed_analysis
# you must first run this command

write_faults fltlist_au.tc -class au.tc 1195
# you must supply a gate ID rather than pin pathname

write_faults fltlist_au.pc -class au.pc mode[3]
# you must supply a pin pathname
```

Example 6

The following example writes out bridge faults in the full format (no *-noeq* switch):

```
write_faults exa.flt
```

```
FaultInformation {
version : 1;
FaultType (Bridge) {
  FaultList {
    Format : Identifier, Class, LocationA, LocationB, BridgeName;
    Instance ("") {
      A0, DS, "/G13", "/G12", "C506";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
      A0, DS, "/G13", "/G12", "C505";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
      A0, DS, "/G13", "/G12", "C504";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
      A0, DS, "/G2", "/G4", "C478";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
    }
  }
}
```

Example 7

The following example writes out bridge faults in an abbreviated format that lists only the net pair and fault classes of each corresponding dominant net:

```
write_faults exa.flt -noeq
```

```
FaultInformation {
version : 1;
FaultType (Bridge) {
  FaultList {
    Format : Identifier, Class, LocationA, LocationB, BridgeName;
    Instance ("") {
      A0, DS, "/G13", "/G12", "";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
      A0, DS, "/G2", "/G4", "";
      A1, DS, -, -, -;
      B0, DS, -, -, -;
      B1, DS, -, -, -;
    }
  }
}
```

Related Topics

add faults

delete faults

[read_fault_sites](#)
[read_faults](#)
[set_checkpointing_options](#)
[report_faults](#)
[set_fault_mode](#)
[set_fault_sampling](#)
[set_fault_type](#)

write_flat_model

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file.

Usage

```
write_flat_model filename
  [-Password password] [-Command_file command_file] [-Replace]
  [-INCLUDE_Proc_simulation_data {Exclude_test_setup_and_test_end | All}]
```

Description

Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file.

Note

 After saving the flat model, you cannot add primary inputs to the model. This includes using the “[add_scan_chains -internal](#)” command.

This command enables you to quickly restore a new session to the state of a previous session based on saved check point data. This command lets you enter into analysis mode without reading the netlist, flattening, and performing DRC.

The write_flat_model command does not save pattern or fault list information in the flat model file. However, you can use the set_checkpointing_options command to save the pattern set and the fault list periodically.

Note

 If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (issuing the create_patterns command) and saving final patterns. For more information, refer to “[Preparing the Design Netlist](#)” in the *Tessent Diagnosis User's Manual*.

When reading in a previously-stored flat model, the tool defaults to the same system mode as when the model was saved. It is also possible to save only the flat model (no scan chain or DRC data) to be restored into setup mode from any system mode.

Arguments

- *filename*

A required string that specifies the name of the file to which you want to write the flattened circuit model.

- -Password *password*

An optional switch and replaceable string pair that directs the tool to embed the specified *password* in the saved file. The *password* is case sensitive and there are no restrictions on its content. Before reading in a flat model that was saved using the -Password switch, the tool will require you to enter the password that was provided with the switch when the model was saved.

For batch runs, you can set the MGC_DFT_FLAT_MODEL_PASSWORD environment variable to the specified password; the tool will then obtain the password from the environment variable instead of querying you for it. If the password setting of this variable is incorrect, however, the tool will revert to querying you for it.

Caution

 Be sure to write down the password and store this copy in a readily accessible, secure location. This will ensure that a forgotten password is only a minor inconvenience.

- -Command_file *command_file*

An optional switch and string pair that specifies the pathname of an ASCII file you previously created that lists commands you want the tool to permanently disable for this flat model. You can create this file with any ASCII text editor.

Choose one of the two keywords, **disable** or **enable**, to place by itself on the first line of the command file. On subsequent lines, list the commands to which the keyword applies, one command or command group per line. The keywords have the following meaning:

- **disable**—Disables the commands listed after the keyword. If the design is one you read into the tool as a flat model, any commands already disabled as a result of the previous flattening will remain disabled; omitting them from the current “disable” list will not re-enable them. All other commands the tool normally supports will be available.
- **enable**—Disables all normally supported commands *except* those listed after the keyword. If the design is one you read into the tool as a flat model, commands already disabled as a result of the previous flattening will remain disabled; listing them after the “enable” keyword will not re-enable them.

Note

 The dofile, exit, and help commands are immune to the command file and are always available.

You specify a command group by listing just the first word or words that all commands in the group have in common. For example, you could list “set” on one line in order to specify the group of all commands that start with the word, “set.”

- -Replace

An optional argument that lets you overwrite an existing circuit model file.

- **-INCLUDE_Proc_simulation_data {Exclude_test_setup_and_test_end | All}**

An optional argument and keyword pair that allows you to save all of the DRC simulation data with the flat model. By default, the tool saves DRC simulation data for everything except for the test_setup and test_end procedures with the flat model. Note that saving all DRC simulation data can save time if you need to debug your test_setup or test_end procedures; however, saving this data can result in a much larger flat model file, especially when the test_setup or test_end procedures contain a large number of cycles.

Examples

Example 1

The following example creates a flat model and saves the model along with all DRC simulation data:

```
create_flat_model
write_flat_model file1 -include_proc_simulation_data all
```

Example 2

The following example uses an external command file to ensure the saved flat model only works with the following commands:

- dofile
- exit
- help
- any command that starts with “set”
- any command that starts with “write_primary”

```
create_flat_model
write_flat_model file1 -command_file commands_for_file1
exit

shell> cat commands_for_file1
enable
set
write primary
```

Related Topics

[diagnose_failures](#)
[create_flat_model](#)
[read_flat_model](#)
[set_checkpointing_options](#)
[write_faults](#)

write_icl

Context: all contexts

Mode: setup, analysis

Prerequisite: ICL modules must be available. You use the [read_icl](#) command to create ICL modules.

Writes out ICL modules created and/or read in with the [read_icl](#) command to the specified file.

Usage

```
write_icl filename [-modules modules_list [-hierarchical]]  
[-exclude_modules exclude_modules] [-created]  
[-modified] [-replace]
```

Description

Writes out ICL modules created and/or read in with the [read_icl](#) command to the specified file.

This command writes out read-in and/or created ICL modules. You can merge both read-in and created ICL modules into the same file. The recommended use model is to write out the ICL hierarchically into a single file which is the default behavior when neither the -modules or -created option are specified.

Use the -modules switch to write only specific ICL modules. The default is to write the top module with all of the modules in the hierarchy below including the read-in modules and the modules below them. Use the -created option if you only want to write out the ICL modules created by ICL extraction and omit those that were read-in

Arguments

- *filename*

A required string that specifies the name of the file to which to write all specified modules.

Note

 For backward compatibility with v2014.4 and earlier releases, you can use the -output_file switch to specify the name of the file. For example:
write_icl -output_file icl_out.icl

- -modules *module_list*

An optional switch and value pair that specifies the names of the modules to write out. The *module_list* value is a Tcl list of one or more ICL module names, or a collection of one or more ICL modules as returned by the [get_icl_modules](#) command.

- -hierarchical

An optional switch that specifies to write out all child modules below the specified ICL modules. This switch can only be used with the -modules switch.

- **-exclude_modules** *exclude_modules*

An optional switch and value pair that excludes ICL modules from being written out. This argument is only useful when the -hierarchical option is inferred or specified. All ICL modules below these excluded ICL modules are also excluded if they are not specified in the -modules list or found below other non-excluded ICL modules.

- **-created**

An optional switch that specifies to limit the written ICL modules to only those *created*.

Specifying this option is equivalent to using the following:

“-modules [get_icl_modules -filter is_created].”

- **-modified**

An optional switch that verifies whether a module has the *is_modified* attribute set to true; if it does, the tool writes out the module.

- **-replace**

An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

Examples

The following example writes out the current design and every other ICL module instantiated below it into a common file.

write_icl icl_out.icl

Related Topics

[get_icl_modules](#)

[read_icl](#)

[set_current_design](#)

write_loops

Context: dft -edt, dft -scan, patterns -scan

Mode: analysis

Writes a list of all the current feedback loops to a file.

Usage

`write_loops filename [-Replace]`

Description

The write_loops command writes all feedback loops in the circuit to a file. For each loop, the file contents show whether the loop was broken by duplication. The file shows loops unbroken by duplication as being broken by a constant value, which means the loop is either a coupling loop or has a single, multiple-fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

Arguments

- *filename*
A required string that specifies the name of the file to which you want to write the loops.
- -Replace
An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example writes a list of all the loops in the circuit to a file:

```
set_system_mode analysis
write_loops loop.info -replace
```

Related Topics

[report_loops](#)

write_memory_repair_dictionary

Context: patterns -ijtag

Mode: analysis

Writes the CompressBisrChain configuration file.

Usage

```
write_memory_repair_dictionary [-file_name file_name]  
    [-return_dictionary]  
    [-fuse_box_controller_icl_instance fuse_box_controller_icl_instance ]  
    [-replace]
```

Description

Writes the configuration file, as a Tcl dictionary, that is required to run the CompressBisrChain script. The resulting file contains all the fuse box controller parameters, as well as the ordered list of BISR register ICL instances listed per power domain group. The CompressBisrChain script is used to program the fuse box using the [FuseBox Access](#) mode, which uses the IJTAG interface instead of using the fuse box controller's autonomous self-fusebox programming mode. For more information, refer to “[Top-Level Verification and Pattern Generation](#)” in *Tessent MemoryBIST User’s Manual For Use with Tessent Shell*.

Arguments

- **-file_name *file_name***
An optional switch and string pair that specifies the name of the file to which you want to write the CompressBisr configuration contents. If this property is not specified, the default output file will be named “BisrCtrlParams.tcl”.
- **-return_dictionary**
An optional switch that instructs the command to return the content as a Tcl dictionary rather than saving it to a file.
- **-fuse_box_controller_icl_instance *fuse_box_controller_icl_instance***
An optional switch and string pair that specifies the fuse box controller ICL instance to use.
- **-replace**
An optional switch that replaces the contents of *file_name* if the file already exists.

write_modelfile

Context: dft -edt, patterns -scan

Mode: analysis

Writes all internal states for a RAM or ROM gate into the file that you specify.

Usage

`write_modelfile filename RAM/ROM_instance_name [-Replace]`

Description

Writes all internal states for a RAM or ROM gate into the file that you specify.

The write_modelfile command writes, in the Mentor Graphics modelfile format, all the internal states for a RAM or ROM gate into a file.

The RAM and ROM internal states are identical to the initial values. If no initial values exist, the command reports an error condition.

Arguments

- ***filename***

A required string that specifies the name of the file to which you want to write the internal states for the RAM or ROM gate. The information is written in Mentor Graphics modelfile format.

- ***RAM/ROM_instance_name***

A required string that specifies the instance name of the RAM or ROM gate whose internal states you want to write. The command reports an error condition if the instance contains multiple RAM/ROM gates.

- **-Replace**

An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example writes all the internal states of a RAM gate into a file for review:

```
add_write_controls 0 w1
set_system_mode analysis
create_patterns
write_modelfile model.ram /p1.ram
```

Related Topics

[read_modelfile](#)

[set_ram_initialization](#)

write_patterns

Context: patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: analysis

Saves the current test pattern set to a file in a specified format.

Usage

For patterns -scan and EDT Off

```
write_patterns pattern_filename
[-pattern_sets [chain] [scan]]
[format_switch]
[-Replace]
[-PROcfile procedure_filename]
[-PARAMeter_file parameter_filename]
[-PARAMETER_List {keyword_value_pair...}]
[-PARAllel | -Serial] [-NOInitialization]
[-BEGin {pattern_number | pattern_name}]
[-END {pattern_number | pattern_name}]
[-TAG tag_name]
[-CELL_placement {Bottom | Top | None}]
[-ENVironment]
[-TEST_Setup {ONE Per file | OFF | ONLY | ONE}]
[-TEST_End {ONE | ONLY | OFF}]
[-EXternal]
[-SAmple [integer]]
[-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-NAme_match | -POSition_match]
[-NOPadding | -PAD0 | -PAD1]
[-MAXloads load_number [-ADD_pattern_index_range_to_filename
| -ADD_Load_range_to_filename]]
[-PATtern_size integer]
[-MEMory_size size_in_KB]
[-SCAN_Memory_size size_in_KB]
[-VERBose]
```

For patterns -scan and EDT On

```
write_patterns pattern_filename
[-pattern_sets [chain] [scan]]
[format_switch]
[-Replace]
[-PROcfile procedure_filename]
[-PARAMeter_file parameter_filename]
[-PARAMETER_List {keyword_value_pair...}]
[-PARAllel | -Serial]
```

[-NOInitialization]
 [-BEgin {*pattern_number* | *pattern_name*}]
 [-END {*pattern_number* | *pattern_name*}]
 [-TAg *tag_name*]
 [-CELL_placement {Bottom | Top | None}]
 [-ENVironment]
 [-TEST_Setup {ONE_Per_file | OFF | ONLY | ONE}]
 [-TEST_End {ONE | ONLY | OFF}]
 [-EXternal]
 [-SAmple [*integer*]]
 [-IDDQ_file]
 [-MODE_Internal | -MODE_External]
 [-PAD0 | -PAD1]
 [-MAXloads *load_number* [-ADD_Pattern_index_range_to_filename
| -ADD_Load_range_to_filename]]
 [-PATtern_size *integer*]
 [-MEMory_size *size_in_KB*]
 [-SCAN_Memory_size *size_in_KB*]
 [{-EDT_Bypass | -EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External]

For patterns -ijtag

write_patterns ***pattern_filename*** [-pattern_sets *pattern_set_list*]
 [format_switch]
 [-Replace]
 [-PROcfile *procedure_filename*]
 [-PARAMETER_file *parameter_filename*]
 [-NOInitialization]
 [-ENVironment]
 [-PARAMeter_List {*keyword_value_pair...*}]
 [-TEST_Setup {ONE_Per_file | OFF | ONLY | ONE}]
 [-TEST_End {ONE | ONLY | OFF}]

For patterns -scan_retargeting

write_patterns ***pattern_filename***
 [-pattern_sets [chain] [scan]]
 [format_switch]
 [-Replace]
 [-PROcfile *procedure_filename*]
 [-PARAMeter_file *parameter_filename*]
 [-PARAMETER_List{*keyword_value_pair...*}]
 [-PARAllel | -Serial]
 [-NOInitialization]
 [-BEgin {*pattern_number* | *pattern_name*}]
 [-END {*pattern_number* | *pattern_name*}]
 [-TAg *tag_name*]
 [-CELL_placement {Bottom | Top | None}]

```
[-ENVironment]
[-TEST_Setup {ONE_Per_file | OFF | ONLY | ONE}]
[-TEST_End {ONE | ONLY | OFF}]
[-EXternal]
[-IDQ_file]
[-MODE_Internal | -MODE_External]
[-NAME_match | -POSition_match]
[-NOPadding | -PAD0 | -PAD1]
[-MAXloads load_number [-ADD_Pattern_index_range_to_filename
    | -ADD_Load_range_to_filename]]
[-PATtern_size integer]
[-MEMory_size size_in_KB]
[-SCAN_Memory_size size_in_KB]
[-VERBose]
```

For Writing Out Hybrid TK/LBIST Flow Parallel Patterns

```
write_patterns pattern_filename -Verilog -PARALLEL -MODE_Internal [-Replace]
```

For Writing Out Hybrid TK/LBIST Flow PatDB Files

```
write_patterns pattern_filename -PATDB
    [-max_scan_load_unload_size integer] [-Replace]
```

For Writing Out User-Defined Broadside Vector Data

```
write_patterns -vector_set vector_set
```

Description

Saves the current test pattern set to a file in a specified format.

By default, the module name created for Verilog test benches consist of the specified design name followed by an underscore, followed by the name of the specified test procedure file followed by _ctl. All periods are converted to underscores. For example:

design_name_test_proc_file_name_ctl

Note that you can type Ctrl-C on the command line to cancel the write_patterns operation without creating any files.

Note

 The tool issues an AG11 DRC error if you attempt to write patterns in a parallel format other than Verilog or STIL using a multiple cycle shift procedure. Any serial format is fine.

By default and when writing patterns, any signal that is classified as unused for the pattern being written is removed from the pattern file, reducing the size of the pattern file or of the Verilog testbench. To disable this behavior, set the [ALL_EXCLUDE_UNUSED](#) keyword to 0.

For techniques to handle slow scan enable transitions, see “[Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions](#)” in the *Tessent Scan and ATPG User's Manual*.

Multiple Processors Use for Writing EDT External Retargeted Patterns

For EDT external patterns, you can use multiple threads to speed-up calculation of output channel values. The supported pattern formats are ASCII, STIL, WGL and Verilog. You must use the [add_processors](#) command and the appropriate number of licenses.

Hybrid TK/LBIST Flow Specifics

Writes out parallel patterns and PatDB files—refer to the [Hybrid TK/LBIST Flow User's Manual](#) for complete information.

IDDQ Fault Type and External Capture Options

When you are using external capture options with IDDQ faults for use with multiple fault types, the tool issues a warning under the following circumstances:

1. You issue the [set_external_capture_options](#) command.
2. You then set the fault type to IDDQ.

Under these circumstances, the tool issues the following warning:

```
// Warning: The set_external_capture_options <option> command is not
// compatible with IDDQ patterns.
// Set_external_capture_options <option> is being ignored while writing
// these patterns.
```

The tool writes the external_mode IDDQ patterns and ignores the external capture options.

Arguments

- *pattern_filename*

A required string that specifies a name for the saved test pattern file. If the filename ends with .gz or .Z, the file is automatically compressed with the GNU gzip utility.

Note

 When you write patterns in the pattern database (PatDB) format, do not end the pattern filename with .gz or .Z, because the PatDB format is already compressed to a size smaller than achievable with the GNU gzip utility. The [read_patterns](#) command does not support .gz or .Z files for the PatDB format.

- *format_switch*

An optional switch that specifies a format for the saved test patterns. You should always save simulation patterns in the same format as the design netlist.

ASIC vendor test data formats usually support only a single timing file. You can specify the test timing that each ASIC vendor requires by using different timing definition files for each format.

Any time you save the pattern set in an ASIC vendor data format, you should also save the patterns in PatDB, ASCII, binary, STIL or WGL format as backup. This is in case you want to read in the patterns from an external file using the [read_patterns](#) command. Only PatDB, ASCII, binary, STIL or WGL formatted test patterns can be read by the [read_patterns](#) command.

Options include:

-Ascii — A switch that writes test patterns in the standard ASCII format. The ASCII format includes the statistics report, environment settings, scan structure definition, scan chain functional test, scan test patterns, and the scan cell information. This is the default.

Excess load and unload values are padded with Xs. You can override this behavior with the **-Nopadding** switch.

If you use the **-External** or the **-Begin** and **-End** switches, test coverage and fault information is omitted from the ASCII pattern set.

-PATDB — An optional switch that directs the tool to write out patterns in a pattern database (PatDB) format.

Use this switch for Hybrid TK/LBIST in LBIST mode, to write patterns as PatDB. The PatDB file includes the pattern information, such as PRPG and MISR values, used when you retarget the core-level PDL using IJTAG to generate top-level patterns.

In ATPG mode, you can use the PatDB format in place of the ASCII format. It is the recommended format for archival patterns. For hierarchical designs, core-level patterns in the PatDB format are the recommended source for retargeting scan patterns to the top level. You can also use them for core-level diagnosis which is performed on the fail logs generated by reverse-mapping chip-level failures.

This format is not available for use in the patterns **-ijtag** context.

-Binary — A switch that writes the test patterns in binary format. This option is not supported in the patterns **-scan_retargeting** context.

-CTI2005 — A switch that writes the patterns in CTL format. The CTL language is used, but in a more structured manner, and with the addition of CTL information. CTL is an extension to the STIL language. You can customize the output format when you utilize a parameter file. For more information, refer to the description of the **-Parameter** switch later in this section and to “[Parameter File Format and Keywords](#)” on page 3937.

-PDL — A switch that writes the pattern back out in PDL format. This format is supported only for patterns **-ijtag** context. The original PDL is converted to iRead and iWrite commands on ScanRegisters and Primary input/output pins. The scan operations needed to perform the ICL network reconfiguration are also explicitly described with iWrite and iApply commands. All PDL events associated with each

pattern set are grouped into individual iProc wrappers and each iProc is named with the name of the corresponding pattern set.

- STIL2005 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0 and 1450.1. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to “[Parameter File Format and Keywords](#)” on page 3937. Note that minimum typing for this switch is -STIL2.
- STil1999 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to “[Parameter File Format and Keywords](#)” on page 3937.
- Verilog — A switch that writes the patterns in Verilog format. The Verilog format contains pattern information and timing information from the timing file as a sequence of events. Saving patterns in Verilog format also causes the tool to write out a configuration (.cfg) file along with the test bench and .vec files. For example:

```
Patterns.v
Patterns.v.0.vec
Patterns.v.1.vec
Patterns.v.cfg
Patterns.v.po.name
Patterns.v.chain.name
```

For information specific to the chain name file see the description of the [SIM_NO_CHAINNAME_FILE](#) parameter keyword.

The configuration file contains the following lines:

```
Patterns.v.0.vec    740
Patterns.v.1.vec    820
```

(The number after the filename is the number of data lines to read from the .vec file.)

Even though the test bench is created with each `write_patterns` command, you only have to compile the design and test bench .v files once. You can use additional patterns just by copying the .vec and .cfg file to the location of the compiled simulation files.

All timeplates that are currently loaded into the tool are also saved in the Verilog test bench, not just the timeplates used by the current set of patterns. If a new procedure file with different timeplates or if the netlist information is modified in any way, you have to compile the new test bench before simulating.

The default Verilog test bench supports the following Verilog plusargs: STARTPAT, ENDPAT, CHAINTEST, END_AFTER_SETUP, SKIP_SETUP, and CONFIG. For more information, refer to “[The Verilog Test Bench](#)” in the *Tessent Scan and ATPG User’s Manual*.

You can use the Verilog patterns to interface to a Verilog simulator such as ModelSim. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to “[Parameter File Format and Keywords](#)” on page 3937.

-Wgl — A switch that writes the patterns in the WGL format. The WGL format contains the waveform pattern information and any timing information from the timing file. You can use the WGL format patterns to generate test patterns in a variety of tester and simulator formats.

-Fjtdl — A switch that writes the test patterns in the Fujitsu ASIC format: FTDL-E.

-MItdl — A switch that writes the test patterns in the Mitsubishi ASIC format: MITDL.

For MITDL format there is restriction that multiple pulse timing must be a cyclical repetition of the first pulse. Consequently, multi-pulse and double-pulse timing in the procedure file only works in the MITDL output without an error if the timing fits the restrictions of the MITDL syntax. See “[Timeplate Definition](#)” in the *Tessent Shell User’s Manual*.

-Mode Lsi — A switch that changes the functionality of the Verilog and WGL output formats so that the saved pattern files meet LSI Logic requirements. This switch is valid only with the -Verilog and -Wgl switches.

-TItdl — A switch that writes the test patterns in the Texas Instruments ASIC format: TDL91. You can customize the output format with the parameter file. For more information on the format of the parameter file, refer to “[Parameter File Format and Keywords](#)” on page 3937.

Note If the design uses a single timeplate the TDL version 4.3 format is used. This removes the TIMING, END_TIMING, and SET_TIMING statements from the test pattern file. Also, the VERSION statement changes from 6.0 to 4.3. However, if the design uses multiple timeplates, then the TDL version output is 6.0.

-TSTL2 — A switch that writes the patterns in Toshiba ASIC format: TSTL2.

-SVf — A switch that writes out the IJTAG patterns in Serial Vector Format. This switch is supported only for “patterns -ijtag” context and can only be used to write IJTAG pattern sets to files.

If the IJTAG patterns make use of more than one TAP interface or of non-TAP scan interfaces, the SVF file will be generated automatically with series of PIO commands instead of SDR, SIR, STATE or TRST commands.

The TCK ratio of the pattern set(s) must be 1 (the TCK ratio is set by means of the -tck_ratio switch of the [open_pattern_set](#) command).

The –test_setup and –test_end switches of the [write_patterns](#) command are taken into account. If requested, the test setup procedure and the test end procedure are also converted to SVF and placed at the beginning/at the end of the pattern file.

- -Replace

An optional switch that replaces the contents of an existing *pattern_filename*.

-
- **-PROcfile procedure_filename**

An optional switch and string that specifies the name of the test procedure file to retrieve non-scan event timing information from. This option saves test patterns using the enhanced pattern output that includes timing information from the test procedure file.

You can specify the procedure filename with this argument or specify it before using this command by issuing the `read_procfile` command. You cannot use this argument with the `-Ascii` or `-Binary` formats.

For more details about the test procedure file, including which output format switches are valid in conjunction with the `-Procfile` switch, refer to “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

- **-pattern_sets [chain] [scan] (patterns -scan only)**

An optional switch and keyword pair that specifies which pattern type (scan or chain) to write from the current session. By default, the tool writes both scan and chain patterns. Note that when you specify both chain and scan patterns, the tool writes chain patterns first regardless of the order of specification.

- **-pattern_sets pattern_set_list (patterns -ijtag only)**

An optional switch and string that specifies a Tcl list of pattern set names that were created using the `open_pattern_set` command. The tool writes patterns in the order specified in the `pattern_set_list`. When you do not use the `-pattern_sets` switch, the command writes all pattern sets in the order specified by the `open_pattern_set` command.

- **-PARAMeter_file parameter_filename**

An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain pattern output formats. For more information about the format of the parameter file, refer to “[Parameter File Format and Keywords](#)” on page 3937.

- **-PARAMETER_List {keyword_value_pair...}**

An optional switch and keyword/value pair that specifies one or more parameter file keywords and their values for use in certain pattern output formats. The keywords and values are delimited by spaces, and you must enclose the list in braces { }. For more information about parameter file keywords, refer to “[Parameter File Format and Keywords](#)” on page 3937.

Note



The `-PARAMETER_List` switch is not supported for ASCII and Binary Pattern formats.

- **-PARAllel**

An optional switch that saves all scan cells in parallel. This is the default.

In designs with scan cells, only scan pins are active during the scan shift cycles. If the tool tries to represent the state of each pin during each shift cycle, it may produce very large pattern files. Simulating the shift operations of these patterns might require a considerable

amount of time if you use a different simulator. You can avoid these problems by saving all scan cells in parallel. The -Parallel switch is compatible with all formats.

Note

 When EDT is On and you save parallel Verilog patterns, the data is saved with respect to the internal scan chains. These patterns are for simulation and cannot be saved as serial. But when you write patterns in parallel tester formats (for example, WGL), the tool saves the data with respect to the external scan channels. These patterns are meant to be saved in serial and cannot be used for parallel simulation.

- -Serial

An optional switch that saves all scan cells in series. You can only use this switch with the -STil1999, -STIL2005, -TItdl (enhanced pattern output only), -Wgl, and -Verilog format type switches.

Note

 You cannot use the -Serial switch and the -Scan_memory_size switch in the same command.

When you use this switch together with the -Verilog switch and the -Parameter switch in the same command, the tool checks the specified parameter file for the SIM_SHIFT_DEBUG keyword. If this keyword exists and is set to 1, every scan chain in the Verilog patterns contain a parallel observe of the entire scan chain for *every shift*, while serially shifting the test patterns. If this keyword is set to 2 or more, the parallel observe occurs for that many shifts, not all. This format is particularly useful for debugging scan chain problems because it allows you to locate the exact scan element causing a problem in a scan chain.

Tip

 Use SIM_SHIFT_DEBUG serial Verilog patterns only for debugging scan chain problems. They typically take more time to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be large). Also, because they simulate serially, the patterns take more time to simulate. Consider using the “-pattern_sets chain,” -Begin, and -End switches to limit the size of the pattern file.

For more information about the parameter file and the SIM_SHIFT_DEBUG keyword, refer to “[Parameter File Format and Keywords](#)” on page 3937.

- -NOInitialization

A switch that writes patterns without creating the initialization cycle in the pattern file. The -Noinitiation switch is valid with the following output formats:

- STil
- SVF
- Verilog
- WG1 (Ascii)

The **-Noinitiation** switch is valid with all enhanced pattern output formats. For more information on the enhanced pattern output, see “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

Note

 If DRC was run with initialization enabled ([set_drc_handling](#) -Initialization_cycle On), which is the default, using the **-Noinitiation** switch when saving patterns may result in simulation mismatches when you verify the patterns in a timing-based simulator.

- **-B**Egin {*pattern_number* | *pattern_name*}

An optional switch and integer (*pattern_number*) or string (*pattern_name*) that specify the pattern to begin storing test patterns. This switch is usually used to specify the first pattern in a range to include in the test patterns. The default *pattern_number* is 0. There is no default *pattern_name*.

For Tessent FastScan and Tessent TestKompress, *pattern_number* integer is assigned to each pattern by the tool during pattern creation. You can use the *pattern_name* string with **-begin** when the patterns have been previously written using the “[write_patterns](#) *pattern_filename* -tag *tag_name*” command in the current session or when you read external patterns with tag names.

After ATPG or pattern retargeting, tag based pattern names are not yet available and you must use **-begin** *pattern_number* to write patterns without tag names or together with the **-tag** *tag_name* to create the *pattern_name*.

When used together with “**-pattern_sets chain**” in the same command, the **-begin** switch applies to chain test patterns only. When used together with “**-pattern_sets -scan**” in the same command, the **-begin** switch applies to scan patterns only.

Only one **-begin** argument can be used per [write_patterns](#) command. To save multiple ranges of patterns, for example patterns 11 through 20 and 81 through 100, use the [set_pattern_filtering](#) command with the **-Range** and/or **-Norange** switches before you issue the [write_patterns](#) command.

If you save only a portion of the internal patterns in the **-Ascii** format, test coverage and fault information is not included.

In this example, the [write_patterns](#) command, using **-begin** *pattern_number*, creates an ASCII format pattern file beginning at pattern 25:

```
create_patterns
write_patterns pat1.ascii -ascii -begin 25
```

This example demonstrates using **-begin** *pattern_name*. The [write_patterns](#) command is used with **-tag** to create the tag name for the patterns and then another [write_patterns](#)

command uses the -begin and -end switches with the tag name to write a subset of the patterns:

```
create_patterns
write_patterns tagged_patterns.ascii -tag my_tag -ascii
write_patterns tagged_pat1.ascii -begin my_tag_25 -end my_tag_49
```

- **-END {pattern_number | pattern_name}**

An optional switch that specifies the last pattern of a range to include in the test patterns. This argument is inclusive; therefore, the tool stores the pattern identified by the pattern_number or pattern_name you specify. The default pattern is the last pattern of the pattern set.

For Tesson FastScan and Tesson TestKompress, pattern_number integer is assigned to each pattern by the tool during pattern creation. You can use the pattern_name string with -begin when the patterns have been previously written using the “write_patterns pattern_filename -tag tag_name” command in the current session or when you read external patterns with tag names.

After ATPG or pattern retargeting, tag based pattern names are not yet available and you must use -end pattern_number to write patterns without tag names or together with the -tag tag_name to create the pattern_name.

When used together with “-pattern_sets chain” in the same command, the -end switch applies to chain test patterns only. When used together with “-pattern_sets -scan in the same command, the -end switch applies to scan patterns only.

If you save only a portion of the internal patterns in the -Ascii format, the tool does not include test coverage and fault information.

In this example, the write_patterns command, using -end pattern_number, creates an ASCII format pattern file ending at pattern 25:

```
create_patterns
write_patterns pat1.ascii -ascii -end 25
```

This example demonstrates using -end pattern_name. The write_patterns command is used with -tag to create the tag name for the patterns and then another write_patterns command uses the -begin and -end switches with the tag name to write a subset of the patterns:

```
create_patterns
write_patterns tagged_patterns.ascii -tag my_tag -ascii
write_patterns tagged_pat1.ascii -begin my_tag_25 -end my_tag_49
```

- **-TAg tag_name**

An optional switch that adds a unique user-specified label, tag_name, to each pattern. All chain tests automatically have “chain as the tag_name” regardless of the tag_name given in the -Tag switch. Since all tag names must be unique, “chain” is not a valid tag_name. If the tag_name supplied is not unique, an error message is issued and the run aborts.

If the tool reads in patterns using the Set Pattern External command, the patterns must also be unique. The run aborts if the tool attempts to make two identically named patterns co-

resident by any method. This uniqueness extends across both the internal and external pattern sets. It is not possible to have the same pattern_name in the internal and external sets.

- **-CELL_placement** Bottom | Top | None

An optional switch and literal pair that controls the placement of the scan cell data in the ASCII pattern file. The literal choices are as follows.

Bottom — A literal that places the scan data after the patterns, at the end of the file. This is the default.

Top — A literal that places the scan data before the patterns, at the top of the file.

None — A literal that excludes the scan data from the file.

- **-ENVironment**

An optional switch that adds information about the current Tesson FastScan or Tesson TestKompress environment to the pattern file as comments. The information includes the identification stamp number, the environment settings, and the DRC rules. The information is the same as the report_environment and report_id_stamp commands display.

- **-TEST_Setup** {ONE_Per_file | OFF | ONLY | ONE}

An optional switch and literal pair that specifies how the command saves the test_setup procedure. If you do not use this switch, the command writes one test_setup procedure at the start of each pattern file.

The literal choices are as follows.

ONE_Per_file — A literal that places one test_setup procedure at the start of each pattern file, even when multiple pattern files are saved with one command using the -maxloads, -memory_size, or -scan_memory_size switches. This is the default option.

OFF — A literal that saves patterns without including the test_setup vectors. Using this option infers the use of the -NOInitialization switch as well. This is because the init vector is not needed in the pattern file when the test_setup vectors are not present.

Note that using this option might produce vectors that do not simulate correctly or might produce failures if used on a tester.

ONLY — A literal that saves the test_setup procedure as a separate file. This option does not work if you specify any of the following switches:

- maxloads
- test_end only
- pattern_sets
- begin
- end

ONE — A literal that applies only one test_setup procedure when both chain and scan test patterns are saved in one pattern file. When saving patterns in formats other than

ASCII and binary, a single test_setup procedure is saved in the first file generated when the -Maxloads switch is used with this “One” option.

- **-TEST_End {ONE | ONLY | OFF}**

An optional switch and literal pair that specifies how the command saves the test_end procedure. If you do not use this switch, one test_end procedure is applied at the end of the pattern file. The literal choices are as follows.

ONE — A literal that applies one test_end procedure at the end of the pattern file after all pattern sets are done. This is the default option. When you also use -maxloads, then the tool saves a test_end procedure at the end of each generated pattern file.

ONLY — A literal that saves the test_end procedure as a separate file. This option does not work if you specify any of the following switches:

- maxloads
- test_end only
- pattern_sets
- begin
- end

OFF — A literal that saves patterns without including the test_end vectors. For ASCII and Binary patterns, this switch has no effect. Note that using this switch might produce vectors that do not simulate correctly or might produce failures if used on a tester.

- **-EXternal**

An optional switch that saves the current external pattern set to the specified *pattern_filename*. By default, internal test patterns are saved.

External patterns saved in ASCII format do not include test coverage and fault information.

- **-SAMple [integer]**

Tip

 This switch has been superseded by the [set_pattern_filtering](#) command, which has multiple sampling options.

An optional switch that specifies to save a sample of patterns for each pattern type instead of saving all the patterns. This can be useful for serial pattern simulation to shorten the simulation time. In Tessent FastScan, the different types are those reported by the [report_statistics](#) command. In Tessent TestKompress, the types are additionally based on whether or not the tool used masking in a pattern. For example, in Tessent TestKompress, a clock-sequential pattern with scan chain masking is considered a different type than a clock-sequential pattern without scan chain masking.

The optional integer specifies the number of patterns contained in each sample. If you do not include an integer, the default is to sample one pattern per type.

This switch is not supported in the patterns -scan_retargeting context.

- **-IDQ_file**

An optional switch that creates an ASCII file that lists the cycle number, vector number, and measure time for each saved pattern that has an IDQ measurement point. This IDQ information file has the same name as the pattern file, with “.IDQ” appended to the end and is created for all enhanced pattern output formats. This switch does not support ASCII or binary output formats.

- **-MODE_Internal**

An optional switch that saves test patterns using the internal mode of the named capture procedures. Test patterns saved with this switch treat all internal clocks and internal primary inputs as directly controllable pins. This is the default when you write patterns in ASCII or binary format, or when you are in LBIST mode in the TK/LBIST hybrid flow.

- **-MODE_External**

An optional switch that specifies to write patterns using the external mode of the named capture procedures and no user-defined internal pins. When setting up for ATPG, if you used the [add_clocks](#) or [add_primary_inputs](#) command to define one or more internal pins, then you need to use this switch when saving patterns. This is the default when you write patterns in tester formats (for example, WGL) and Verilog format.

LBIST mode in the TK/LBIST hybrid flow does not support external mode patterns.

Note

If a capture procedure does not contain an external mode, the -Mode_external switch uses the internal mode cycle information, but ignores internal PI events. The switch also ignores internal pin events in scan procedures (shift, load_unload, and so on).

- **-Lib *work_dir* (EDT Off)**

An optional switch and string pair that specifies the working directory containing the compiled top level design. The default work_dir is “work” in the current directory.

- **-NAmE_match (EDT Off)**

An optional switch that instantiates the device under test in the test bench using the name matching method. If you start with a Verilog netlist and do not restart with a flat model, the Verilog test bench uses the position matching method.

- **-POsition_match (EDT Off)**

An optional switch that places the device under test in the test bench using the position matching method.

- **-NOPadding (EDT Off)**

An optional switch that prevents the tool from padding shorter scan chains with X values for the input data in saved ASCII test patterns. When there are scan chains of differing lengths in the design, the tool pads the patterns for the shorter chains with X values to represent unknown or *don't care* bits in the input data. Use this switch to eliminate X values in the input data due to short scan chains. This switch can only be used for ASCII test patterns.

This switch cannot be used with EDT patterns.

To load unpadded test patterns into a tool, use the [read_patterns](#) command.

- **-PAD0 | -PAD1**

For EDT Off

Optional switches that replace X values, representing unknown or *don't care* bits, with either a 1 or a 0 value in the saved test patterns. Options include:

-PAD0 — Replaces X values with 0 values.

-PAD1 — Replaces X values with 1 values.

This behavior is the same for STIL, WGL, and Verilog. When you do not specify -pad0 or -pad0, an X, which is “do not care”, is used on shorter chains. For parallel load scan loading, it is also left as X. For serial loads in a Verilog test bench, the tool shifts in each bit one vector at a time so it has access to the value of the previous bit and updates the X with the force value from the previous vector. Since padding with X means “do not care”, it is correct to use the previous force value in the place of that X.

This switch is only valid for test patterns that support scan chain padding.

For EDT On

Optional switches that replace X values with binary values in the saved test patterns. This switch only affects the unload values in the test patterns. The binary value may be a 0 or 1 and cannot be specified. Both -PAD0 and -PAD1 operate the same on Tessent TestKompress test patterns.

- **-MAXloads *load_number***

An optional switch and positive integer pair that specifies the number of scan loads, *load_number*, to include in a pattern file. Much like the -Begin and -End switches, the -Maxloads switch automatically breaks up the pattern file into a series of pattern files with the specified number of scan loads in each. For example, suppose pattern creation results in 730 basic scan patterns. The following command uses -Maxloads,

```
write_patterns my_patterns -maxloads 100
```

saves the same sequence of files with the same pattern content as this series of commands:

```
write_patterns my_patterns_1 -begin 0 -end 99
write_patterns my_patterns_2 -begin 100 -end 199
...
write_patterns my_patterns_7 -begin 600 -end 699
write_patterns my_patterns_8 -begin 700
```

In either case, the first file contains 100 chain and scan patterns, every other file would contain 100 scan patterns; the last file would contain the remaining patterns.

When there are fewer chain test patterns than the specified number of maxloads, the first file may contain fewer scan patterns than the specified number.

You can also use -Maxloads to break up a pattern subset you specify with the -Begin and/or -End arguments in the same command. For example, here is the command to write patterns 50 through 99 to a sequence of files that each contain no more than 15 patterns:

```
write_patterns my_patterns -maxloads 15 -begin 50 -end 99
```

When you use -Maxloads, the tool automatically appends an underscore and sequence number to the name of each resultant pattern file, starting with number one (_1). When you have more chain test patterns than maxloads, the tool issues this warning:

```
// Warning: The number of chain test patterns, 39, exceeds the
//          number of scan loads specified for a single file,
//          the first file will have extra scan loads.
//          Use a larger value for -maxloads, -memory_size, or
//          -scan_memory_size to correct this.
```

Note

 This switch is only valid with ASCII, Binary, and enhanced pattern output and cannot be used with the -Pattern_size switch.

- **-ADD_Pattern_index_range_to_filename**

A boolean option that adds the pattern index range to the split pattern filename for the files that are generated with the -maxloads option. When specified, the tool appends an underscore and the pattern index range (_0_999, _1000_1999) to the filename, rather than an underscore and sequence number. The default for this option is false.

The filenames reflect the range of patterns in the file. The filename has this pattern: temp_0_8.stil. Each file should have a maxloads number of patterns except the last.

The first file created contains a chain test as well as the scan test.

- **-ADD_Load_range_to_filename**

A boolean option that adds the range of loads to the split pattern filename for the files that are generated with the -maxloads option. When specified, the tool appends an underscore and the load range (_0_999, _1000_1999) to the filename, rather than an underscore and sequence number. The default for this option is false.

The filenames reflect the range of loads in the file. The filename has this pattern: temp_0_8.stil.

The first file created contains a chain test as well as the scan test.

- **-PATtern_size *integer***

An optional switch and integer pair which specifies the size of the memory buffer and pattern file in which to save. *Integer* is given in kilobytes.

Note

 The -Pattern_size switch is valid only when using -Verilog output.

The default pattern size is 128MB. However, any size specified for a pattern size will be adjusted to hold a multiple of the largest pattern. For example, if the largest Tessent FastScan or Tessent TestKompress pattern requires 3MB for one pattern, then the file size will be a multiple of 3MB, which would result in a 126MB pattern size. The maximum pattern size is 2147483646KB (approximately 268GB), and the tool issues an error if this size is exceeded.

- **-MEMory_size *size_in_KB***

An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of available main tester memory, and sets this number as a limit on the size of the pattern data in a pattern file. Much like the -Maxloads switch, the -Memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes (KBs) of pattern data. The tool assumes that one pattern value (0, 1, X, or Z) requires one byte on the tester.

If you specify this switch and the -Scan_memory_size switch, the -Memory_size switch specifies the KB limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

Note that the number of vectors in each pattern file will vary. This is because each file contains the vectors from the test_setup procedure and all the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

- **-SCAN_Memory_size *size_in_KB***

An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of tester memory available for only scan pattern data, and sets this number as a limit on the size of this data in the pattern file. Much like the -Maxloads switch, the -Scan_memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes (KBs) of scan pattern data. The tool assumes one scan pattern value (0, 1, X, or Z) requires one byte on the tester.

If you specify this switch and the -Memory_size switch, the latter specifies the KB limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

Note

 You cannot use the -Scan_memory_size switch and the -Serial switch in the same command.

Note that the number of vectors in each pattern file varies; this difference occurs because each file contains the vectors from the test_setup procedure and the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

- **-VERBose (EDT Off)**

An optional switch that writes extra information messages about the files being saved to stdout or the tool's log file. These messages use the following syntax:

```
// Writing file_type file_name
```

Where *file_type* is a phrase describing what the file is, and *file_name* is the specified or generated file name. For example, *file_type* could be “STIL patterns,” “WGL patterns,” “Verilog testbench,” “Verilog vectors,” and so forth.

- [{-EDT_Bypass | - EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External] (EDT On)

Optional switch that specifies how EDT test patterns are saved. By default, Tessent TestKompress (EDT on) saves ASCII test patterns that define load data with respect to the input channels of the EDT logic and unload data with respect to the core scan chains. Options include:

-EDT_Bypass — An optional switch used with the -BInary format switch to save EDT bypass test patterns in binary format. EDT bypass test patterns are the decompressed version of the EDT test patterns. These options are only valid for EDT test pattern generation. For more information, see “[Generation of Identical EDT and Bypass Test Patterns](#)” in the *Tessent TestKompress User’s Manual*.

Note

 This option cannot be used when bypass chains are inserted in the design netlist before the EDT logic is generated.

-EDT_Single_bypass_chain — An optional switch used with the -BInary format switch to save EDT bypass test patterns for the single bypass chain configuration. The test patterns are saved in binary format. These options are only valid for EDT test pattern generation. For more information, see “[Dual Bypass Configurations](#)” in the *Tessent TestKompress User’s Manual*.

-EDT_Internal — An optional switch that saves EDT test pattern data for only the internal core scan chains to an ASCII file. The test patterns contain core scan chain force and observe data except for X values observed on the output of the compactor due to X blocking or scan chain masking. For more information, see “[Understanding Scan Chain Masking in the Compactor](#)” in the *Tessent TestKompress User’s Manual*.

Note

 Test patterns saved with the -Edt_internal switch cannot be read back into the tool.

-EDT_External — An optional switch that saves EDT test pattern data with respect to the (compressed) input and output channels. These test patterns are similar to WGL and STIL tester formats and must be resimulated before they can be saved out to an external test pattern file.

- -max_scan_load_unload_size *integer*

An optional switch and value pair that specifies the number of LBIST patterns that will have their scan chain data stored in the pattern database for diagnostic purposes. This option does not affect the PRPG/MISR/low-power seeds/signatures; the pattern database continues to store them for the entire pattern range. The default is 1024.

- -vector_set *vector_set*

A switch and value pair that specifies a vector set specified as the pattern set to write.

This switch is mutually exclusive with any other write_patterns command switch or option. See “[Vector Data Callbacks](#)” in the *Tessent Scan and ATPG User’s Manual*.

By design, when using vector_callback procs and writing patterns, the data fed to the vector callback procs must be the internal view pattern data, and therefore, any external_mode cycles from a named capture, or external_mode cycles from PLL_cycles or external_capture are not used. When you enable vector callbacks, the tool issues the following warnings:

```
// Warning: The set_write_pattern_options -vector_callback command
// causes write_patterns to use the internal_mode view of the
// pattern data. Set_external_capture_options -PLL_cycles is ignored
// in that case.
```

Or:

```
// Warning: The set_write_pattern_options -vector_callback command
// causes write_patterns to use the internal_mode view of the
// pattern data. Set_external_capture_options -capture_proc is
// ignored in that case.
```

Examples

Example 1

The following example performs an ATPG run, and then saves only the first 15 test patterns to a file in the Verilog format, including the timing information contained in the timing file:

```
set_system_mode analysis
create_patterns
write_patterns file1 -verilog timefile -end 14
```

Example 2

The following example illustrates the module name created in the enhanced Verilog output when using the -Procfile switch. If the design name is MAIN, and you issue the following write_patterns command:

```
write_patterns pattern1.pat -procfile -verilog -replace
```

the module name in the test bench will be MAIN_pattern1_pat_ctl.

Example 3

The following example (in patterns -scan) saves Verilog chain test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the chain test patterns (for debugging scan chain problems). The example assumes the parameter file, test.param, contains a “SIM_SHIFT_DEBUG 1;” line.

```
write_patterns chain_shift_debug_patt.v -verilog -pattern_sets chain -serial \
-parameter test.param
```

Example 4

The following examples illustrate the output of the -Verbose switch:

```
write_patterns test_pat.v -verilog -replace -verbose
```

[write_patterns](#)

```
// Writing Verilog testbench test_pat.v
// Writing Verilog PO names test_pat.v.po.name
// Writing Verilog chain names test_pat.v.chain.name
// Writing Verilog vectors test_pat.v.0.vec
// Writing Verilog vectors test_pat.v.1.vec

write_patterns test_pat.stil -stil -replace -verbose -maxloads 10

// Writing STIL patterns test_pat_0.stil
// Writing STIL patterns test_pat_1.stil
```

Example 5

The following example writes the test_setup procedure as a separate STIL file:

write_patterns test_setup.stil -stil -replace -test_setup only

Example 6

This example uses the -add_pattern_index_range_to_filename switch to add the pattern range to the split pattern filename. The maxloads is 10 and the pattern count is 29. Three files are created: temp_0_8.stil, temp_9_18.stil, and temp_19_28.stil. The first file only has 9 scan patterns, instead of 10, because it contains one chain test pattern.

```
write_patterns temp.stil -stil -verbose -maxloads 10 -add_pattern_index_range -replace

// command: write_patterns temp.stil -stil -verbose -maxloads 10
//           -add_pattern_index_range_to_filename -replace
// Writing STIL pattern          temp_0_8.stil
// // Writing STIL pattern        temp_9_18.stil
// Writing STIL pattern          temp_19_28.stil
```

Related Topics

[add_scan_groups](#)
[analyze_simulation_mismatches](#)
[get_write_patterns_options](#)
[read_patterns](#)
[read_procfile](#)
[report_measure_cycles](#)
[report_procedures](#)
[report_timeplate](#)
[report_write_patterns_options](#)
[set_external_simulator](#)
[set_pattern_classification](#)
[set_write_patterns_options](#)
[write_procfile](#)

Vector Creation and Modification

write_primary_inputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Writes primary inputs to the specified file.

Usage

```
write_primary_inputs filename [-Replace] [-All | primary_input_pin...]
```

Description

Writes primary inputs to the specified file.

The write_primary_inputs command writes a list of either all the primary inputs of a circuit or a specific list of primary inputs that you specify into a file where it can be reviewed.

This command is identical to the [report_primary_inputs](#) command except the data is written into a file.

Arguments

- ***filename***
A required string that specifies the name of the file to which the tool writes the primary inputs.
- **-Replace**
An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.
- **-All**
An optional switch that specifies to write all the primary inputs to the file. This is the default.
- ***primary_input_pin***
An optional repeatable string that specifies a list of primary input pins that you want to write to the file.

Examples

The following example writes all primary inputs to a file:

```
write_primary_inputs inputfile
```

Related Topics

[report_primary_inputs](#)

write_primary_outputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Writes primary outputs to the specified file.

Usage

`write_primary_outputs filename [-Replace] [-All | primary_output_pin...]`

Description

Writes primary outputs to the specified file.

The `write_primary_outputs` command writes a list of either all the primary outputs of a circuit or a specific list of primary outputs that you specify into a file where it can be reviewed.

This command is identical to the [report_primary_outputs](#) command except the data is written into a file.

Arguments

- ***filename***

A required string that specifies the name of the file to which the tool writes the primary outputs.

- **-Replace**

An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

- **-All**

An optional switch that specifies to write all the primary outputs to the file. This is the default.

- ***primary_input_pin***

An optional repeatable string that specifies a list of primary output pins that you want to write to the file.

Examples

The following example writes all primary outputs to a file:

`write_primary_outputs outputfile`

Related Topics

[report_primary_outputs](#)

write_procedure_testbench

Context: dft -edt, patterns -scan

Mode: analysis

Writes a test bench for verifying correct operation of internal signals used in procedures.

Usage

```
write_procedure_testbench testbench_filename [-Replace]  
[-PARAMeter parameter_filename] [-AL1] | -CApture capture_name... | -LOad_unload]
```

Description

Writes a test bench for verifying correct operation of internal signals used in procedures.

The write_procedure_testbench command writes a Verilog test bench for verifying correct operation of internal signals used in procedures, including named capture procedures. The test bench will verify that if a procedure forces or pulses an internal signal, the value forced or pulsed is actually produced by the circuit when simulated in a Verilog simulator. (The expect values in the test bench are based on the statements in the procedure file that force or pulse internal pins.)

Note

 The test bench written by this command is unrelated to the test bench written by the [write_patterns](#) command, which enables you to simulate and verify the ATPG patterns. Saving patterns is the last step of the ATPG process, whereas you should write a test bench to verify named capture procedures before you create patterns.

Determining from incorrect patterns where the mistakes are in the procedures can be difficult. This command helps you find the mistakes in the procedures, so you can correct them *before* they result in incorrect patterns.

Arguments

- ***testbench_filename***

A required string that specifies the name of the file to which you want to write the test bench.

- -Replace

An optional switch that replaces the contents of *testbench_filename* if the file already exists. By default, the command will not overwrite an existing file.

- -PARAMeter *parameter_filename*

An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain output formats. For more information on the format of the parameter file, refer to the “[Parameter File Format and Keywords](#)” on page 3937.

- **-All**

An optional switch that specifies to write a test bench that simulates the test_setup procedure, followed by all named capture procedures simulated as load_unload/capture pairs. Values forced on internal signals in all procedures are used as expected responses. Named capture procedures that do not force values on internal signals are skipped. An error message is issued if none of the procedures force values on internal signals. This is the default.

- **-CAPture *capture_name***

An optional switch and repeatable string that specifies a list of named capture procedures to verify. In this case, the command will write a test bench that simulates the test_setup procedure, followed by the load_unload procedure with shift procedure, and then the specified named capture procedure(s). If you specify more than one named capture procedure, the test bench simulates the additional ones as additional load_unload/capture pairs. Each load_unload/capture pair is simulated twice. Values forced on internal signals in the named capture procedures are used as expected responses. An error message is issued if a specified named capture procedure does not force any values on internal signals.

- **-LOAD_unload**

An optional switch that specifies to write a test bench that simulates the load_unload and shift procedures. This test bench simulates the test_setup procedure (if present), followed by the load_unload procedure with one shift. Values forced on internal signals in the load_unload and shift procedures are used as expected responses.

Examples

Example 1

The following example writes a Verilog test bench for verifying the functionality of the named capture procedure “cap1.”

```
write_procedure_testbench verilog_tb_cap1.v -capture cap1 -replace
```

Example 2

The following example assumes the pathname of the ModelSim bin directory has previously been added to the PATH variable. From the shell, the example first creates a Verilog library named “proc_test_work,” then compiles the following three elements into the library:

- Source for the top level design
- Test bench written out in the preceding example
- Parts library

With these prerequisites met, the example simulates the test bench.

```
% vlib proc_test_work
% vlog -work proc_test_work counter_syn_scan.v verilog_tb_cap1.v \
      -v my_verilog_source/verilog_lib.v
```

```

Model Technology ModelSim SE vlog 6.1 Compiler 2005.06 Jun  6 2005
-- Compiling module mux21_macro
-- Compiling module sffr_macro
-- Compiling module lmmux
...
-- Scanning library file 'my_verilog_source/verilog_lib.v
-- Compiling UDP dff_r_err
-- Compiling UDP dff_r

Top level modules:
    counter_verilog_tb_cap1_v_ctl

vsim counter_verilog_tb_cap1_v_ctl -lib proc_test_work \
    -t 100ps -c -voptargs=+acc=npr -do "run -all"

...
# //
# Loading proc_test_work.counter_verilog_tb_cap1_v_ctl
# Loading proc_test_work.counter
# Loading proc_test_work.count4
...
# Loading proc_test_work.dff_r
# Loading proc_test_work.dff_err
# Loading proc_test_work.dff_p
# run -all
#           1850: inst_0/QI_nx7 simulated 0  expected 1
#           1850: Procedure: CAPTURE1 Event: force inst_0/QI_nx7 to 1 at 0

```

The last two lines indicate the procedure named “capture1” forces a 1 on the internal pin /inst_0/QI_nx7, but the value produced on the pin when simulated in the Verilog simulator is 0. This

suggests the force statement is applying an incorrect value. The next example shows what the corresponding force statement (highlighted in bold) might look like in the test procedure file:

```
procedure capture capture1 =
    scan_group groupcnt ;
    mode internal =
        timeplate gen_tp1 ;
        cycle =
            force clear 0 ;
            force clock[0] 0 ;
            ...
            force clock[9] 0 ;
            force test_clk 0 ;
            force /inst_0/QI_nx7 1;
            force_pi ;
            measure_po ;
            pulse clock[0] ;
            ...
            pulse clock[4] ;
    end;
    mode external =
        timeplate gen_tp1_long ;
        cycle =
            force clear 0 ;
            force clock[0] 0 ;
            ...
            force clock[9] 0 ;
            force test_clk 0 ;
            force_pi ;
            measure_po ;
            pulse clock[0] ;
            ...
            pulse clock[4] ;
    end;
end; // capture1
```

Example 3

The following example shows what the message at the end of the Verilog simulation run looks like if there are no mismatches:

```
# run -all
# No error between simulated and expected patterns.
```

Related Topics

[report_capture_procedures](#)
[report_procedures](#)
[report_timeplate](#)
[set_capture_procedures](#)

write_procfile

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Writes existing procedure and timing data to the named test procedure file.

Usage

`write_procfile proc_filename [-Replace] [-Full [-EXternal]] [-EXpand_iprocs]`

Description

Writes existing procedure and timing data to the named test procedure file.

For complete information on test procedure file creation, syntax, and structure, refer to “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

The write_procfile command writes out existing procedure and timing data to the named test procedure file.

Arguments

- ***proc_filename***

A required string that specifies the name of the file to which you want to write existing procedure and timing data.

- **-Replace**

An optional switch that replaces the contents of the file if the *proc_filename* already exists.

- **-Full**

An optional switch that causes the tool to parse the ATPG pattern list (if any) and create all needed non-scan procedures before writing the procedure file data. In Tessent FastScan and Tessent TestKompress, this option will always create a capture procedure.

Note

 The -Full option can cause the write_procfile command to take more time if there are a large number of ATPG-generated patterns in the internal pattern list.

- **-EXternal**

An optional switch that causes the -Full switch to use the external patterns specified by the read_patterns command instead of internally generated patterns. This switch behavior is similar to the -External switch behavior for the write_patterns command. The -External switch is only valid when using the -Full switch.

- **-EXpand_iprocs**

An optional switch that replaces all iCall and iMerge statements within procedures with the cycles that represent the solution to these iCall statement. For iMerges, the tool replaces the

entire iMerge with the cycles representing the solution to that iMerge. The start and end of the expanded cycles is marked by comments in the reported procedure or procedure file. Use this switch to when debugging [E14](#) DRC violations.

Examples

The following example writes the existing procedure and timing data to the specified file:

```
write_procfile myprocfile.proc
```

Related Topics

[add_scan_groups](#)

[read_procfile](#)

[report_procedures](#)

[report_timeplate](#)

[write_patterns](#)

write_scan_order

Context: dft -scan

Mode: insertion

Writes the scan chains into the SCANCHAINS section of the ScanDEF file, where each statement describes a single scan chain. ScanDEF is a subset of the DEF language.

Usage

```
write_scan_order filename [-Replace] [-use_escaping_rule {Verilog | Lefdef}]
```

Description

The write_scan_order command writes the scan chains into the SCANCHAINS section of the ScanDEF file, where each statement describes a single scan chain. ScanDEF is a subset of the DEF language. When the tool successfully traces through a subchain, the cells that comprise that subchain are written to the FLOATING list of the corresponding scan chain's statement by default. In this case, the order of cells comprising such a subchain doesn't have to be preserved, and cells can appear in the START and STOP sections.

Modules which have the “exclude_from_concatenated_netlist” attribute specified are considered as hard macros and are not included in the concatenated netlist. Moreover, when writing the scanDEF file, any traced scan elements from existing segments on these modules are omitted between the scanIn and scanOut sections of these segments, and stop at the SI/SO boundary, as is done for a cell library.

Arguments

- ***filename***

A required string that specifies the name of the DEF file.

- **-Replace**

An optional switch that replaces the specified DEF file, if it exists.

- **-use_escaping_rule {Verilog | Lefdef}**

An optional switch and literal pair that specifies whether to use the Verilog escaping mechanism or the LEF/DEF escaping mechanism for specifying special characters in strings/identifiers. The default is Verilog.

In the Verilog escaping mechanism, you start the escaped identifier with a backslash character, “\”, and terminate the escaped identifier with a white space. Neither the leading backslash character nor the terminating white space is considered to be part of the identifier.

In the LEF/DEF escaping mechanism, you use the backslash character, “\”, in front of any character that has a special meaning in LEF/DEF, in which case the special meaning of the character is ignored.

Note

 The escape character cannot be used with the pound sign, '#', in LEF/DEF.

Examples**Example 1**

This example illustrates the information reported with the report_scan_cells command and written to the scanDEF file.

report_scan_cells

Clock	Chain	Group	ShiftReg		Library	Clock		
	CellNo	Name	Name	Pathname	ID/CellNo	ModelName	ScanOut	Pinname
	Polarity							
0	chain1	dummy	/ud5	-/-	sff	QB	clk1	(+)
1	chain1	dummy	/ud4	1/4	dff	QB	clk1	(+)
2	chain1	dummy	/ud3	1/3	dff	Q	clk1	(+)
3	chain1	dummy	/ud2	1/2	dff	Q	clk1	(+)
4	chain1	dummy	/ud1	1/1	sff	Q	clk1	(+)
5	chain1	dummy	/ud10	3/2	dff	Q	clk1	(+)
6	chain1	dummy	/ud9	3/1	sff	Q	clk1	(+)
7	chain1	dummy	/ud8	-/-	sff	QB	clk1	(+)
0	chain2	dummy	/ud11	-/-	sff	Q	clk1	(+)
1	chain2	dummy	/ud7	2/2	dff	QB	clk1	(+)
2	chain2	dummy	/ud6	2/1	sff	Q	clk1	(+)
3	chain2	dummy	/ud12	-/-	sff	Q	clk1	(+)
4	chain2	dummy	/ud13	-/-	sff	Q	clk1	(+)
5	chain2	dummy	/ud14	-/-	sff	Q	clk1	(+)
6	chain2	dummy	/ud15	-/-	sff	Q	clk1	(+)

The tool identifies three shift registers as shown in the following output generated by the report_shift_registers command.

report_shift_registers -verb

Id	Length	Path	Hierarchical	SequentialCell	Clock	Library
			InstanceName	Edge & Name	ModelName	
[1]	4	/	ud1	+ clk1	sff	
			ud2	+ clk1	dff	
			ud3	+ clk1	dff	
			ud4	+ clk1	dff	
[2]	2	/	ud6	+ clk1	sff	
			ud7	+ clk1	dff	
[3]	2	/	ud9	+ clk1	sff	

Given this input, the following command generates the scan DEF file that follows.

write_scan_order results/scandef -replace

Note that in the following file, regular flip-flops and shift register flip-flops within the same domain are separated by FLOATING and ORDERED lists, respectively. Each shift register is written out in a separate ORDERED list.

```

#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 2 ;

- chain1_sub0
+ START ud8 QB
+ ORDERED
    ud9 ( IN SI ) ( OUT Q )
    ud10 ( IN D ) ( OUT Q )
+ ORDERED
    ud1 ( IN SI ) ( OUT Q )
    ud2 ( IN D ) ( OUT Q )
    ud3 ( IN D ) ( OUT Q )
    ud4 ( IN D ) ( OUT QB )
+ STOP ud5 SI

# Partition for core chain in clock clk1 (pos-edge) domain
+ PARTITION partition_1 MAXBITS 6 ;

- chain2_sub0
+ START ud15 Q
+ FLOATING
    ud14 ( IN SI ) ( OUT Q )
    ud13 ( IN SI ) ( OUT Q )
    ud12 ( IN SI ) ( OUT Q )
+ ORDERED
    ud6 ( IN SI ) ( OUT Q )
    ud7 ( IN D ) ( OUT QB )
+ STOP ud11 SI

# Partition for core chain in clock clk1 (pos-edge) domain
+ PARTITION partition_1 MAXBITS 5 ;

END SCANCHAINS

END DESIGN

```

Similarly, given the same input, the following command generates the scan DEF file that follows:

`write_scan_order scan.def`

Note the following about the scan DEF file:

- sub1 and sub2 are instances of library cell libsubchmodel, which contains a subchain of length 2. The scan DEF includes this length in the BITS field. For regular cells of length 1, the BITS field is not printed.
- All flip-flops are written out in one ORDERED list because chain1 contains cells in different clock edge domains.

```

VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]";
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1
+ START u5 Q
+ ORDERED
  u1 ( IN SI ) ( OUT Q )
  u2 ( IN D ) ( OUT Q )
  u3 ( IN D ) ( OUT Q )
  u4 ( IN D ) ( OUT Q )
  u6 ( IN SI ) ( OUT Q )
  u7 ( IN D ) ( OUT Q )
  u8 ( IN SI ) ( OUT Q )
  u9 ( IN D ) ( OUT Q )
  u10 ( IN D ) ( OUT Q )
  u11 ( IN SI ) ( OUT Q )
  u12 ( IN D ) ( OUT Q )
  u13 ( IN SI ) ( OUT Q )
  u14 ( IN D ) ( OUT Q )
  u15 ( IN SI ) ( OUT Q )
  lckup1 ( IN D ) ( OUT Q )
  sub1 ( IN sci1 ) ( OUT scol ) ( BITS 2 )
+ STOP sub2 sci2 ;

END SCANCHAINS

```

Example 2

This example illustrates the information reported with the `report_scan_cells` command and written to the scanDEF file when traceable subchains are present in the design.

`report_scan_cells`

CellNo	Chain	Group	CellName	ScanOut	Clock	Clock
	CellName	Name	Pathname			
	Polarity					
-	chain1	dummy	/lckup1	latch	Q	clk2 (-)
0	chain1 (schi2)	dummy	/uB/f3	sff	Q	clk2 (+)
1	chain1 (schi2)	dummy	/uB/f2	sff	Q	clk2 (+)
2	chain1 (schi2)	dummy	/uB/f1	sff	Q	clk2 (+)
-	chain1	dummy	/lckup2	latch	Q	clk2 (+)
0	chain2 (schc1)	dummy	/uWA/uA/f21	sff	Q	clk (+)
1	chain2	dummy	/ud	sff	Q	clk (+)
2	chain2	dummy	/us	sff	Q	clk (+)
0	chain3 (scho1)	dummy	/uWA/uA/f31	sff	Q	clk (+)
-	chain4	dummy	/lckup3	latch	Q	clk (-)
0	chain4 (schi1)	dummy	/uWA/uA/f3	sff	Q	clk (+)
1	chain4 (schi1)	dummy	/uWA/uA/bb/f2	sff	Q	clk (+)
2	chain4 (schi1)	dummy	/uWA/uA/f1	sff	Q	clk (+)
-	chain4	dummy	/lckup4	latch	Q	clk (+)

write_scan_order

```

#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]";
DESIGN top ;
UNITS DISTANCE MICRONS 1000 ;
SCANCHAINS 1 ;

- chain1_sub0
+ START lckup2 Q
+ FLOATING
  uB/f1 ( IN SI ) ( OUT Q )
  uB/f2 ( IN SI ) ( OUT Q )
  uB/f3 ( IN SI ) ( OUT Q )
+ STOP lckup1 D

# Partition for core chain in clock clk2 (pos-edge) domain
+ PARTITION partition_1 MAXBITS 3 ;

- chain2_sub0
+ START us Q
+ FLOATING
  ud ( IN SI ) ( OUT Q )
+ STOP uWA/uA/f21 SI

# Partition for core chain in clock clk (pos-edge) domain
+ PARTITION partition_2 MAXBITS 1 ;
# The following chain segment with only 1 or 2 scan cells has been
# commented out for compatibility with the layout tools.
#- chain3_sub0
# + START uWA/uA/f31 SI
# + STOP uWA/uA/f31 Q ;
- chain4_sub0
+ START lckup4 Q
+ FLOATING
  uWA/mux ( IN A1 ) ( OUT Y ) ( BITS 0 )
  uWA/uA/f1 ( IN SI ) ( OUT Q )
  uWA/uA/bb/f2 ( IN SI ) ( OUT Q )
  uWA/uA/f3 ( IN SI ) ( OUT Q )
+ STOP lckup3 D

# Partition for core chain in clock clk (pos-edge) domain
+ PARTITION partition_2 MAXBITS 3 ;

END SCANCHAINS

END DESIGN

```

Example 3

The following example is a modification of previous example. In this example, the [set_attribute_value](#) command assigns the attribute `is_hard_module` to the module `blackBox`. An instance of this module, `uWA/uA/bb`, is written to the scanDEF file unexpanded and the number of scan cells on its scan path appears in the `BITS` statement.

```

set_attribute_value blackBox -name is_hard_module -value true

#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1_sub0
+ START lckup2 Q
+ ORDERED
    uB/f1 ( IN SI ) ( OUT Q )
    uB/f2 ( IN SI ) ( OUT Q )
    uB/f3 ( IN SI ) ( OUT Q )
+ STOP lckup1 D

# Partition for core chain in clock clk2 (pos-edge) domain
+ PARTITION partition_1 MAXBITS 3 ;

- chain2_sub0
+ START us Q
+ FLOATING
    ud ( IN SI ) ( OUT Q )
+ STOP uWA/uA/f21 SI

# Partition for core chain in clock clk (pos-edge) domain
+ PARTITION partition_2 MAXBITS 1 ;

# The following chain segment with only 1 or 2 scan cells has been
# commented out for compatibility with the layout tools.
# - chain3_sub0
# + START uWA/uA/f31 SI
# + STOP uWA/uA/f31 Q ;

- chain4_sub0
+ START lckup4 Q
+ ORDERED
    uWA/mux ( IN A1 ) ( OUT Y ) ( BITS 0 )
    uWA/uA/f1 ( IN SI ) ( OUT Q )
    uWA/uA/bb ( IN si ) ( OUT so ) ( BITS 1 )
    uWA/uA/f3 ( IN SI ) ( OUT Q )
+ STOP lckup3 D

# Partition for core chain in clock clk (pos-edge) domain
+ PARTITION partition_2 MAXBITS 3 ;

END SCANCHAINS

END DESIGN

```

Example 4

The following example shows command output when power domains are present in the design:

write_scan_order

```
- chain1_sub0
+ START I2/df_0003_1 QB
+ FLOATING
    I2/df_0002_1 ( IN SI ) ( OUT QB )
+ STOP I2/df_0001_1 SI
# Partition for core chain in clk (pos-edge) clock domain of
# PD2 power domain
+ PARTITION partition_1 MAXBITS 1 ;

- chain2_sub0
+ START I10/df_0002_1 QB
+ FLOATING
    I10/df_0001_1 ( IN SI ) ( OUT QB )
+ STOP I10/df_0003_1 SI
# Partition for core chain in clk (pos-edge) clock domain of
# PD3 power domain
+ PARTITION partition_2 MAXBITS 1 ;
```

write_scan_setup

Context: dft -test_points

Mode: insertion

Writes out a dofile containing all necessary steps for scan insertion.

Usage

```
write_scan_setup -prefix filename_prefix [-replace] [-all_internal_clocks]
```

Description

Writes out a dofile containing all necessary steps for scan insertion.

The existing chain and sub-chain information will be carried over from previous dofiles.

When you use this command for test point insertion in a post-scan design, the tool writes out the dofile with the test point flops. The name of the new dofile is *filename_prefix_tp_flops*.

Arguments

- **-prefix *filename_prefix***

A required switch and string that specifies a name prefix for the scan setup files.

The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the *filename_prefix*, and the files are written to that location.

- **-replace**

An optional switch that replaces the contents of each file that already exists.

- **-all_internal_clocks**

An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Examples

The following example writes out a dofile and test procedure file that can be used from the dft - scan context to perform x-bounding and scan insertion on the design:

```
analyze_test_points  
insert_test_logic  
write_design -output netlist_with_test_points.v -replace  
write_scan_setup -prefix scan_setup -replace
```

Related Topics

[analyze_test_points](#)

[insert_test_logic](#)

[report_test_points](#)
[set_test_point_types](#)
[set_test_point_analysis_options](#)
[set_test_point_insertion_options](#)
[write_scan_setup](#)
[write_test_point_dofile](#)

write_test_point_dofile

Context: dft -test_points

Mode: all modes

Writes out a dofile containing the test points.

Usage

```
write_test_point_dofile filename [-all | -control | -observe] [-tool_analyzed_only] [-replace]
```

Description

Writes out a dofile containing the test points.

Writes out a dofile consisting of [add_control_points](#) and/or [add_observe_points](#) commands that describe the currently identified test point locations. This dofile can be used to manually edit the list of test points and override the default selections prior to inserting the hardware. It can also be used to completely bypass the analysis step during a subsequent tool invocation.

Arguments

- *filename*

A required string that specifies the name of the file into which you want to write the test points information.

- -all | -control | -observe

An optional switch that specifies the type of test point to write to the dofile. Choose one from the following:

- all — Writes out both control and observe points.

- control — Writes out only the control points.

- observe — Writes out only the observe points.

- -tool_analyzed_only

An optional switch that writes out only the tool generated only test points to the dofile. If you do not specify this switch, then the tool writes out both the tool-generated and the manually-added test points.

- replace

An optional switch that directs the tool to overwrite an existing *filename*.

Examples

The following example performs test point analysis and then writes out the dofile:

```
analyze_test_points
```

```
write_test_point_dofile my_test_points.dofile -replace
```

Related Topics

[add_control_points](#)
[add_observe_points](#)
[analyze_test_points](#)
[insert_test_logic](#)
[report_test_points](#)
[set_test_point_analysis_options](#)
[set_test_point_insertion_options](#)
[set_test_point_types](#)
[write_scan_setup](#)

write_tsdb_data

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Populates the contents of the Tesson Shell Data Base (TSDB).

Usage

```
write_tsdb_data [-design_id string] [-pattern_id string]
                 [-max_scan_load_unload_size integer] [-replace]
```

Description

Writes the contents of the **Tesson Shell Data Base (TSDB)**, which is a structured directory containing subdirectories and files. The subdirectory naming is designed to group related information for each specific view of each design block. The files in the TSDB include the flat model, the Tesson Core Description file, the PatDB pattern file, and the fault list.

Note

 For different configurations, you must use different modes, that is each fault type and each configuration requires a different mode. For example, if you have high compression and low compression patterns, you must generate these patterns using different modes. Refer to the [set_current_mode](#) command.

When using this command, the fault list is not written out if pattern filtering is enabled with the [set_pattern_filtering](#) command.

LogicBIST Usage

Writes the `<design_name>.lbist_mode[<mode_name>]` directories that contain all of the files needed to enable logicBIST test mode during [PatternsSpecification](#) processing and that allow diagnosis of the failures once in the production flow. The files generated after LBIST fault simulation are used when you use IJTAG to generate the top-level patterns.

The command detects that it has to create the `<design_name>.lbist_mode[<mode_name>]` directory when the context is patterns -scan and at least one [add_lfsrs](#) command was specified in the setup mode. You can check whether the logicBIST mode is enabled using the “[get_context_logic_bist](#)” command.

ATPG and Pattern Retargeting Usage

Writes the `<design_name>.atpg_mode[<mode_name>]` or `<design_name>.atpg_retargeting_mode[<mode_name>]` directories. When performing ATPG and pattern retargeting in Tesson Shell, the following usage applies:

- If you invoke `write_tsdb_data` after core-level ATPG, the tools writes the files you specified. These files are needed for retargeting core-level patterns, as well as the

patterns and the flat model needed for core-level diagnosis after reverse-mapping of chip-level failures.

- The TCD for the core will be auto-loaded (if not already in memory) at the next level when performing pattern retargeting and [add_core_instances](#) is used to add the design core.
- For “patterns –scan_retargeting” context and performing core extraction, using `write_tsdb_data` writes out the TCD. No flat model/patterns/fault list are written out in this case. If it is chip level, this TCD will be used for reverse-mapping of fails back to the cores, or for subsequently performing pattern retargeting without rerunning core extraction. See “[Scan Pattern Retargeting](#)” in the *Tessent Scan and ATPG User's Manual*.

Arguments

- `-design_id string`

An optional switch and value pair that specifies the *design_id* value to use when creating the [logic_test_cores](#) TSDB sub-directory.

If you specify the *design_id* with the `-design_id` switch, then the tool uses this *design_id*. A *design_id* must only use a combination of the following characters:

- Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
- Numeric. For example: “123”.
- Underscore. For example: “A_1”.

If you do not specify the *design_id*, then the tool uses the value of the [design_id](#) module attribute that is set on the current design module when this attribute has a non-null value. When you load a design with the [read_design](#) command, then the tool sets the *design_id* attribute when reading the design. Additionally, the tool also sets this attribute after elaboration of the current design to the value returned by [get_context -design_id](#) when the value of the *design_id* is null (the design was not loaded with `read_design`, but, rather, with [read_verilog](#)). If the *design_id* module attribute on the current design is still null by the time you use the `write_tsdb_data` command, then the tool uses the value returned by [get_context -design_id](#).

Do not confuse *design_id* with *mode_name* and *pattern_id* when creating the [logic_test_core](#) containers. The mode name is set using the [set_current_mode](#) command while in the setup mode of the -patterns scan context. The *pattern_id* is specified using the `-pattern_id` switch and is used to differentiate different pattern sets for a given mode name of a given design view. The *design_id* defines the design view used; this is typically used to differentiate the pre-layout versus post-layout design views so that you can create your post-layout patterns without losing the pre-layout copies.

- **-pattern_id *pattern_id***

An optional switch and value pair that specifies the *pattern_id* when saving the <design_name>_<mode_name>_<fault_type>[_<pattern_id>].faults.gz file. For a specific fault type, you may have performed different simulations such as a different value of n-detect. For UDFM, you may have a different set of faults during different fault simulation runs. The optional pattern_id is a way to save the fault classification under different file names.

If you do not specify a -pattern_id string, the *pattern_id* portion of the file name is omitted.

- **-max_scan_load_unload_size *integer***

An optional switch and value pair that specifies the number of LBIST patterns that will have their scan chain data stored in the pattern database for diagnostic purposes. This option does not affect the PRPG/MISR/low-power seeds/signatures; the pattern database continues to store them for the entire pattern range. The default is 1024.

- **-replace**

An optional switch and value pair that suppresses the error that is normally generated when you rerun the command and the <design_name>[_<design_id>].logic_test_core (see “[logic_test_cores](#)” on page 3776) directory already exist. Instead of producing an error, the files inside the <design_name>[_<design_id>].logic_test_core directory are silently overwritten. Notice that if you issue the command multiple time with different pattern_id values, the files that are independent of the pattern id will exist because they were saved by the last invocation include the following:<design_name>_<mode_name>.flat.gz, <design_name>_<mode_name>.patdb, <design_name>_<mode_name>.tcd.gz files.

Multiple <design_name>_<mode_name>_<fault_type>[_<pattern_id>].faults.gz files can coexist in the directory.

Examples

In this example, the write_tsdb_data command is invoked after fault simulation is performed to store all of the files that will be needed to generate signoff and manufacturing patterns and perform diagnosis once the chip is in production. The symbol <...> represents the path “./tsdb_outdir/logic_test_cores/mychip_pre_layout.logic_test_core”. The “_pre_layout” string is appended to the design name in the “mychip_pre_layout.logic_test_core” name because the -design_id option was specified when the [set_context](#) command was issued.

```
set_context patterns -scan -design_id pre_layout
<read_verilog and setup commands here>set_current_mode normal

set_system_mode analysis
set_fault_type stuck
add_faults -all
set_random_patterns 100

simulate_patterns -source bist -store_patterns all
write_tsdb_data -replace
```

```
// Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.tcd.gz  
// Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.flat.gz  
// Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.patdb  
// Writing <...>/mychip.lbist_mode_normal/m8051_lbist_stuck.faults.gz
```

Related Topics

[write_core_description](#)
[write_patterns](#)
[write_flat_model](#)
[write_faults](#)

write_visualizer_dofile

Context: unspecified, all contexts

Mode: setup, analysis

Writes a dofile containing the commands needed to recreate the current instance and data sets displayed in the DFTVisualizer windows.

Usage

`write_visualizer_dofile file_pathname [-Display window_name...] [-Replace]`

Description

Writes a dofile containing the commands needed to recreate the current instance and data sets displayed in the DFTVisualizer windows.

The commands in this dofile do not replicate the entire command sequence you have used in the tool session, but rather they simply add the instances and data sets currently present in the specified displays. For example, if you add ten instances to the Hierarchical Schematic window, delete three of them, then issue the `write_visualizer_dofile` command, the resultant dofile will contain commands that add seven instances to the Hierarchical Schematic window.

Arguments

- ***filename***

A required string that specifies the name of the dofile you are writing.

- **-Display *window_name***

An optional switch and repeatable literal that specify the DFTVisualizer windows to which the dofile should restore current instances and data sets. The valid options for the *window_name* argument, from which you can select as many as you want, are as follows:

`FLAt_schematic` — A literal that specifies the Flat Schematic window.

`HIEarchical_schematic` — A literal that specifies the Hierarchical Schematic window.

`DAta` — A literal that specifies the Data window.

`Browser` — A literal that specifies the Design Browser window.

`Signals` — A literal that specifies the Signals window.

Omit the `-Display` switch to include commands in the dofile that restore current instance and data displays in the following windows: Data, Flat and Hierarchical Schematic.

- **-Replace**

An optional switch that specifies to replace the contents of *file_pathname* if a file by that name already exists.

Examples

The following example writes a dofile that will restore instances and data sets currently displayed in the Flat and Hierarchical Schematic windows of DFTVisualizer:

```
write_visualizer_dofile debug.do -display flat_schematic hierarchical_schematic -replace
```

Related Topics

[close_visualizer](#)

[open_visualizer](#)

[set_visualizer_preferences](#)

write_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Writes the current DFTVisualizer preference settings to the specified file.

Usage

`write_visualizer_preferences filename`

Description

Writes the current DFTVisualizer preference settings to the specified file.

To read in the preference settings, use the [read_visualizer_preferences](#) command.

Arguments

- *filename*

A required string that specifies the name of the text file to which to write the current DFTVisualizer preference settings.

Examples

The following example writes the current DFTVisualizer preference settings to the file, *my_prefs.dftvis*:

```
set_system_mode analysis
write_visualizer_preferences my_prefs.dftvis
```

Related Topics

[read_visualizer_preferences](#)

write_window_contents

Context: unspecified, all contexts

Mode: setup, analysis

Saves the current content of the specified DFTVisualizer window to the specified file.

Usage

```
write_window_contents file.pathname [-Replace] -Display window_name  
[-Format file_format]
```

Description

Saves the current content of the specified DFTVisualizer window to the specified file.

The window must be visible, not minimized. You can save the file in one of several formats, both graphic and text formats.

To view a graphic file generated by this command, you can use any third-party tool that supports that particular graphic format.

Arguments

- ***file.pathname***

A required string that specifies the pathname of the screen capture file.

- **-Display *window_name***

A required switch and literal pair that specifies the DFTVisualizer window whose content you are capturing. Select the window name from the following list:

FLAt_schematic — A literal that specifies the Flat Schematic window. This is the invocation default.

HIEarchical_schematic — A literal that specifies the Hierarchical Schematic window.

Browser — A literal that specifies the Design Browser window.

Signal — A literal that specifies the Signals window.

DAta — A literal that specifies the Data window.

WAve — A literal that specifies the Wave window.

TRanscript — A literal that specifies the Transcript window.

Format — A literal that specifies the Format window.

SEarch — A literal that specifies the Global Search window.

TEst_structures — A literal that specifies the Test Structures window.

- **-Replace**

An optional switch that specifies replacement of the contents of *file.pathname*, if a file by that name already exists.

- -Format *file format*

An optional switch and literal pair that specify the format of the capture content. By default, JPEG format is used. Format options include:

schematic — A literal that specifies the Mentor Graphics Tessent schematic format.
This is the default for the Flat and Hierarchical Schematic windows.

PS — A literal that specifies the Postscript format. This format can only be used with the schematic window.

JPG (or JPEG) — A literal that specifies the JPG file interchange format. This is the default for all windows except the Flat and Hierarchical Schematic windows.

GIF — A literal that specifies the Graphics Interchange Format.

PNG — A literal that specifies the Portable Network Graphics format.

TXT — A literal that specifies ASCII text format. This option is valid for the Design Browser, Data, and Signal windows only.

CSV — A literal that specifies the CSV (comma-separated values) file format. This option is valid for the Design Browser, Data, and Global Search windows.

Examples

The following example analyzes a DRC violation (automatically invoking DFTVisualizer in the process and displaying the part of the netlist related to the violation in the Flat Schematic window), then saves a screen capture of the displayed netlist segment in GIF format:

```
analyze_drcViolation c3-1
// command: open_visualizer -display flat_schematic
// Note: Gate report now set to clock_cone (clock=/tck).
// Creating Schematic for 5 instances

write_window_contents my_c3-1.gif -display flat_schematic -format gif
// INFO: "flat schematic window" contents saved in my_c3-1.gif
// 10599 bytes) using the "gif" format.
```

Related Topics

[close_visualizer](#)
[open_visualizer](#)
[read_window_contents](#)
[set_visualizer_preferences](#)

Chapter 7

Design Rule Checking

Design rule checks are separated into categories based on the type of verification they perform.

Design Rule Overview	2640
Scan Insertion Checking	2640
How to Troubleshoot Rules Violations	2641
Design Rules	2653
RAM Rules (A Rules)	2666
Assertion Rules (ASSERT Rules)	2680
BIST Rules (B Rules)	2686
Clock Rules (C Rules)	2687
Scan Cell Data Rules (D Rules)	2732
Pre-DFT Clock Rules (DFT_C Rules)	2747
Extra Rules (E Rules)	2775
EDT Finder Rules (F Rules)	2796
Flattening Rules (FN, FP, and FG Rules)	2821
General Rules (G Rules)	2829
ICL Extraction Rules (I Rules)	2834
ICL Semantic Rules (ICL Rules)	2843
EDT Rules (K Rules)	2895
Procedure Rules (P Rules)	2916
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)	2961
Scannability Rules (S Rules)	2984
Scan Chain Trace Rules (T Rules)	2994
Power-Aware Rules (V Rules)	3018
Timing Rules (W Rules)	3032
Other DRC Messages	3052

Design Rule Overview

The design rules are organized into functional categories.

Scan Insertion Checking	2640
How to Troubleshoot Rules Violations	2641
Rules Settings	2641
How to Turn on ATPG Analysis	2641
How to Debug DRC Violations in DFTVisualizer.....	2641
How to Set the Level of Gate Data	2646
How to Set the Gate Information Type.....	2648
How to Report Gate Data	2648
Flattening Rule Violations	2652
Design Rule Output	2652

Scan Insertion Checking

Prior to scan insertion, the tool performs limited rule checks on the design as you switch from setup to analysis mode. Part of the checking it does is scannability checking.

For more information, refer to [Scannability Rules \(S Rules\)](#).

For primary clock inputs gated by other logic, a test procedure file describes the logic conditions that permit propagation of the clock signal through these gates. For uncontrollable clock circuitry, the tool can assist you in modifying your circuit by inserting test logic circuitry at these clock nodes whenever necessary. Refer to “[Test Logic Insertion](#)” in the *Tessent Scan and ATPG User’s Manual* for details.

If you specify existing scan circuitry, or if you have a test procedure file that sets up conditions to allow some state elements to be scan candidates, the tool performs more extensive checking. After you add scan circuitry to your design and generate or write a test procedure file, you should go back to setup mode and specify this information. Then you can return to analysis mode and perform extensive rules checking *before* entering patterns -scan context.

How to Troubleshoot Rules Violations

You will need to troubleshoot your design to identify the cause of design rule violations.

Rules Settings	2641
How to Turn on ATPG Analysis	2641
How to Debug DRC Violations in DFTVisualizer	2641
How to Set the Level of Gate Data	2646
How to Set the Gate Information Type	2648
How to Report Gate Data	2648
Flattening Rule Violations.....	2652
Design Rule Output	2652

Rules Settings

Some rules allow you to specify how a violation is handled either error, warning, note, or ignore.

How to Turn on ATPG Analysis

The Atpg_analysis option to the set_drc_handling command provides full test generation analysis during rules checking for clock rules C1, C3, C4, C5, C6, some D rules and some E rules (such as E4, E5, E8, E10, E11 and E12). For the C1 rule, this analysis is enabled by default; for other rules, the tool does not perform the analysis unless you direct it to.

For example, assume that you select Atpg_analysis for clock rule C1 and the tool simulates a clock input to be X. The rule violation occurs when the analysis determines that it is, or may be, possible to turn the clock input on when all other defined clocks are off, and constrained pins are at their constrained values.

Note

 The ATPG analysis rules checking process may require additional CPU time and memory.

How to Debug DRC Violations in DFTVisualizer

When you issue the analyze_drcViolation command, the tool automatically opens the relevant DFTVisualizer schematic window (if it is not already open) and sets gate reporting to match the type of rule violation that has occurred.

In the schematic window, DFTVisualizer displays *callouts* next to failing gates or clocks associated with DRC violations. Each callout displays a summary of the DRC and potential next

steps. For more information about using callouts, see “[Flat Schematic Window](#)” in the *Tessent Shell User’s Manual*.

[Table 7-1](#) lists the DRCs that are supported by callouts in the DFTVisualizer schematic windows and their messages and appearance.

Table 7-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
A1	Failing RAM instance	Setting all write controls off failed to force off RAM write line. Trace write control lines backward to debug the issue further.
A2	Failing RAM instance	Connected to scan clock.
A3	Failing RAM instance	Read or write control line is connected to address input of RAM. Trace address input backwards to debug the issue further.
A4	Failing RAM instance	Read or write control line is connected to data input of RAM. Trace data input backward to debug the issue further.
A6	Failing RAM instance	Write/Read line not off during time <time> of <test proc name> procedure. Refer to the test procedure file to debug the problem further.
A7	Failing RAM instance	Setting all read controls off failed to force off RAM read line. Trace read control lines backward to debug the issue further.
C3	Source of violation	Clocked by same clock/clock_path.
	Failing instance	Captures data from sequential element <source_gate> clocked by same clock </clock_path>.
C4	Source of violation	Clocked by same clock/clock_path.
	Failing instance	Captures data from sequential element <source_gate> clocked by same clock </clock_path>.
C5	Failing instance	Multiple clock inputs of <clock names> are in the same clock cone and the clock inputs may be on at the same time. Trace back on the clock cone to debug the problem further.
C6	Failing instance	Clock <clock_path> affects both data and clock input.
C7	Failing instance	Clock input </clock_path> cannot capture data with a single clock on.
C8	Primary output pin	Primary output <PO name> is connected to clock <clock name>.

Table 7-1. DRC Violation Messages (cont.)

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
C9	Primary output pin	Path from clock <clock name> is gated by scan cell that uses same clock.
C16	Failing instance	Load value of scan cell is disturbed by clock input <input_num> at the beginning of the capture phase when pin constraints are forced.
DFT_C1	A primary input is reached	<p>When stopped on a port, the callout is “add_clock -period missing for this port.” The error message is:</p> <pre>// Error: The memory clock pin 'blockB_clka_i1/CLK' // (63.14) is sourced by port 'clkb' (2.0) but its // period was // not defined using the 'add_clocks -period' command. // (DFT_C1-2).</pre> <p>When stopped on a pin, the callout is “Clock tracing blocked here, and no ‘add_clock -reference’ or ‘add_clock -period’ was defined.” The error message is:</p> <pre>// Error: The memory clock pin 'blockA_clka_i1/CLK' //(26.16) source tracing stop at pin 'clock_mux1/Y' (84.0) // Correct the block conditions or use the 'add_clocks // -period' if this is an embedded oscillator or // 'add_clocks -reference' if this an active PLL or clock // divider. (DFT_C1-1)</pre>
DFT_C2	The pin or port on which the trace stopped	Not declared as a clock
DFT_C3	The pin or port on which the trace stopped	Not declared as a clock
DFT_C4	The pin on which the clock is defined	(derived) clock source
DFT_C5	The clock with the missing period	No period specified
D1	Failing instance	Scan cell disturbed during time <time> procname.
D2	Failing copy instance	Copy cell failed data capture design rule check. See Scan Cell Data Rules (D Rules) for conditions that cause this DRC violation.

Table 7-1. DRC Violation Messages (cont.)

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
D3	Failing Master FF	Not successfully observed by master observe procedure. Refer to the test procedure file to debug the problem further.
D4	Failing Master FF	Skew load procedure not successful for MASTER. Refer to the test procedure file to debug the problem further.
D5	Failing FF	Non-scan memory element converted to <gate type>: TIEX, TIE0, TIE1, INIT-0, INIT-1, INIT-X, and TLA.
D6	Non-scan latch instance	Latch not transparent due to <reason for failure>.
D7	Failing instance	Clock port set to stable high at the end of shift procedure.
D8	Failing Master latch	Master latch allows data capture while clocks off. Refer to the test procedure file to debug the problem further.
D9	Failing FF	<Type of disturbance> disturbance in procedure <sequential transparent proc name>. Refer to the test procedure file to debug the problem further.
D10	Failing FF	Invalid transparency (X/Y) in clock procedure <clock proc name>.
E7	Failing bidirectional pin	Bidirectional pin <pin name> not set to input mode at time <time> of <test proc name> procedure. Refer to the test procedure file to debug the problem further.
F2	Failing gate	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F5	Failing gate	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
	Failing channel input	Channel input is not driving any decompressor.
F6	1st Failing state element	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
	2nd Failing state element	Channel input is not driving any decompressor.
F7	Failing state element	Driven by a primary input which is not defined as a channel input in the dofile.
	Channel input	Not defined as a channel input in the dofile, but drives a decompressor. Refer to the dofile or the netlist to debug the problem.

Table 7-1. DRC Violation Messages (cont.)

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
F8	Channel out port	The channel output is not driven by a compactor. Refer to the netlist or trace the schematic to debug the problem further.
	Channel in port	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F9	Failing state element	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F12	Scan in pin	Scan in pin of the uncompressed scan chain was not defined using “add scan chains” command.
	Scan out pin	Scan out pin of the uncompressed scan chain was not defined using “add scan chains” command.
F13	Failing state element	State element is not reset correctly prior to shift. Refer to the netlist or test procedure file to debug the problem further. Simulated Value = Expected Value =
F14	Failing state element	State element does not hold its value during capture. Refer to the netlist or test procedure file to debug the problem further.
F15	Failing state elements in the path	Initialized to wrong value <val> after test_setup and load_unload. Expected value is <val>. Refer to the netlist or test procedure file to debug the problem further.
F16	Failing state element	State element does not preserve its value during capture and load_unload. Refer to the netlist or test procedure file to debug the problem further.
F17	Failing state element	Failing state elements of the MISR.
F18	Failing Mask hold register	State element does not load expected values mask shift register before shift in load_unload. Refer to the netlist or test procedure file to debug the problem further. Simulated Value = <val> Expected Value = <val>
	2nd Failing state element	The driving mask shift register.
F19	Failing state elements	Does not preserve its value during shift. Refer to the netlist or test procedure file to debug the problem further.

Table 7-1. DRC Violation Messages (cont.)

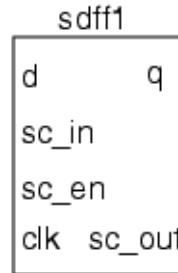
DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
I1	Failing instance	Ports found in ICL module are not found on corresponding design module.
I2	Failing instance	ICL port does not have a valid direct path to or from another ICL port function.
I3	Failing instance	ICL port is connected to an incompatible ICL port function.
I4	Failing instance	ICL port is connected to a primary input or output port previously inferred to be an incompatible ICL port function.
I5	Failing instance	Input constraint verification failed for an input pin on an ICL-attributed design instance.
P92	Failing gate	Internal primary input <path> is not controlled.
R2	Scan input pin	Tracing back from this scan input pin/scan_input_path could not reach a top-level port.
	Top-level gate	Could not trace further back from this gate.
R4	Failing pin	This pin had an input constraint of expected value (core module_name mode internal/external) but currently has value (actual Val).
R5	Failing instance	The stable_after_setup value of pin was expected val but currently has an actual value.
R8	Failing gate	Last traced gate.
V8	Scan cell	Power Domain: < name >, Power Mode (CPF) or Power State: <name>.
V9	Scan cell	Power Domain: < name >, Power Mode (CPF) or Power State: <name>.
V13	1st failing gate	Power Domains <power_domain_name>.
	2nd failing gate	Power Domains <power_domain_name>.
V14	1st failing gate	Power Domains <power_domain_name>.
	2nd failing gate	Power Domains <power_domain_name>.

How to Set the Level of Gate Data

The tools can report data at different levels; therefore, you should specify the level of information before you issue the `report_gates` command.

You do this with the `set_gate_level` command. Setting the gate level to design (the default) reports information at the design cell (library model) level. The following figure depicts a scannable-equivalent DFF cell library model at the design level.

Figure 7-1. Example of Design Level



If the gate level is set to design, the `report_gates` command displays the following information:

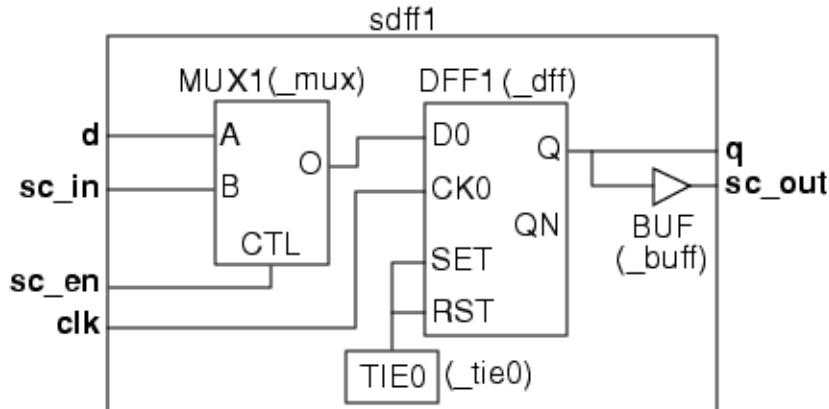
```

instance_name      cell_type
  input_pin_name   I  (data)    pin_pathname ...
  input_pin_name   I  (data)    pin_pathname ...
.
  input_pin_name   0  (data)    pin_pathname ...

```

Setting the gate level to primitive reports information at the simulation gate level. The following figure depicts the `sdff1` library model at the primitive level.

Figure 7-2. Example of Primitive Level



If you set the gate level to primitive, the `report_gates` command displays the following information:

```

instance_name      (gate_id#)    gate_type
  input_pin_name   I  (data)    gate_id#-pin_pathname ...
  input_pin_name   I  (data)    gate_id#-pin_pathname ...
.
  input_pin_name   0  (data)    gate_id#-pin_pathname...

```

In the DFTVisualizer Flat Schematic window, you can change between the Primitive and Design levels by pressing **Ctrl + L**, or by selecting **Display > Gate Level (Ctrl + L) > Design | Primitive**.

How to Set the Gate Information Type

The `set_gate_report` command specifies the type of information that you want to appear when you report_gates data with the `report_gates` command. The multitude of options this command supports varies somewhat depending on which tool you are using.

The common usage of the `set_gate_report` command is as follows:

```
set_gate_report {Normal | Trace | Error_pattern | TlE_value  
| [Drc_pattern procedure_name {time | -All}] {-COnstrain_value [ON | OFF]}}
```

- The Trace option displays the values of the gates obtained during scan chain tracing. That is, this option displays data obtained on an error condition (not warning) during simulation of the **shift** procedure. You can use this option to help determine why a scan chain was not properly sensitized.
- The Error_pattern option displays the simulated values of the gate and its inputs, for the pattern (event) that had an error. This option displays such information as cell disturbances during the **load_unload** procedure or bus contention problems.
- The Normal option is the default. It displays only standard connectivity data.
- The Drc_pattern option displays an identified procedure's simulated gate values during the designated time. This option is similar to the Trace option, but is more versatile because it allows access to the data obtained from simulation of any of the test procedures.
- The Parallel_pattern option displays simulated values for a selected pattern in the last simulation pass. A “pattern” is any time event that occurs during the test procedure. When the ATPG tool encounters problems in generating patterns, you can access the simulation data with this option.
- The -COnstrain_value {ON | OFF} option, when on, displays the simulated values that result from all naturel tied gates, learned constant value non-scan cells, constrained pins, and constrained cells, the blocked flag (B), and the unused flag (UU).

How to Report Gate Data

If you encounter rules violations when you attempt to exit the Setup mode, you may need more information about specific gates in the design for troubleshooting purposes. While the violation message may give some information as to the location of the problem, you may need to track down the source of the problem by reporting on a sequence of gates in the design. The `report_gates` command is a very powerful command you can use to report on netlist data.

The following subsections show how to use report_gates to display various types of information for troubleshooting purposes. For more information on this command, refer to the report_gates reference page.

In DFTVisualizer, you can report data on selected gates by clicking the  Report Gates icon, pressing **Ctrl + R**, or pressing the right mouse button and selecting **Commands > Report Gates**. For more information, see “[Reporting Gates](#)” in the *Tessent Shell User’s Manual*.

How to Report on a Specific Gate

You can use the report_gates command to display information for selected gates, which you identify by either a gate index number or a pin pathname of a pin connected to the gate. This command reports the gate name, its gate type, and its connectivity to other gates. For example, to use report_gates in this manner, you could specify:

```
report_gates 74493
```

[Figure 7-3](#) shows a report with primitive-level information for a gate with an ID number of 74493.

Figure 7-3. Data Reported for a Specific Gate

Instance Name	Gate ID#	Learned Behavior (Inactive High Latch)
<pre>rep ga 74493 // /b5/u12.u1_0_M (74493) LA-IH // "S I (000) 11426- // "R I (000) 6694- // "C0 I (000) 36060- // d I (XXX) 53753-/b2/u4/Y // scnck I (010) 28049-/b5/BOS595/CK2 // sd I (XXX) 11775-/b5/u12.u1_1_S/q // "OUT O (XXX) 11427- // MASTER cell_id=0 chain+c1 group=g1 invert_data=FFFF</pre>	<pre>// /b5/u12.u1_0_M (74493) LA-IH // "S I (000) 11426- // "R I (000) 6694- // "C0 I (000) 36060- // d I (XXX) 53753-/b2/u4/Y // scnck I (010) 28049-/b5/BOS595/CK2 // sd I (XXX) 11775-/b5/u12.u1_1_S/q // "OUT O (XXX) 11427- // MASTER cell_id=0 chain+c1 group=g1 invert_data=FFFF</pre>	<pre>Connectivity Data</pre>
<pre>"S I (000) "R I (000) "C0 I (000) d I (XXX) scnck I (010) sd I (XXX) "OUT O (XXX)</pre>	<pre>I (000) I (000) I (000) I (XXX) I (010) I (XXX) O (XXX)</pre>	<pre>11426- 6694- 36060- 53753- 28049- 11775- 11427-</pre>
<pre>Pin Names</pre>	<pre>Pin Types</pre>	<pre>Pin Data</pre>
<pre>cell_id=0 chain+c1 group=g1 invert_data=FFFF</pre>		
<pre>Scan Chain Data</pre>		

How to Report on All Gates of a Specified Type

You can use report_gates to report on all gates of a specified type. The report_gates usage for this case is:

```
report_gates {-Type gate_type}...
```

The supported gate types are those listed as simulation primitives in the “[Simulation Primitives of the Flattened Model](#)” section of the *Tessent Scan and ATPG User’s Manual*.

The following example shows how to report on all TIE0 gates.

```
report_gates -type tie0

// -----
// List of TIE0 gates
// -----
// /u1/inst_565_ff_d_0_dff (13) TIE0
//   "OUT" O 267- 266-
// /u1/inst_565_ff_d_1_13 (14) TIE0
//   "OUT" O 269- 268-
// /u1/inst_565_ff_d_2_13 (15) TIE0
//   "OUT" O 271- 270-
// Total number of tie0 gates = 3
```

How to Report a Histogram of All Gate Types

You can use report_gates to show a distribution (histogram) of all gates in the design. To use report_gates in this manner, specify:

report_gates -type Histogram

The following example shows the type of data this command displays.

```
// -----
// List of histogram of gates
// -----
// BUF=175 INV=30 AND=3 NAND=17 OR=7 NOR=5 XOR=2 LA=14
// PI=12 PO=7 TIE0=7 MUX=7
```

How to Report on a Path Between Two Gates

You can also use report_gates to display information on the circuitry between two specified gates. To use report_gates in this manner, specify:

report_gates -path <gate1_ID#> <gate2_ID#>

How to Report on the First Input of a Gate

The report_gates command can display data on the gate connected to the first input of the previously reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

back

The following example shows how to use report_gates and back to trace backward through the first input of the previously reported gate.

report_gates 26

```
// /u1/inst_565_ff_d_1_13 (26) BUF
//   "I0" I 269-
//   "OUT" O 268- 75-
```

back

```
// /u1/inst_565_ff_d_1_13 (269) LA
// "S" I 14-
// "R" I 145-
// SCLK I 4-/clk
// D I 265-/u1/_g32/X
// ACLK I 2-/scan_mclk
// SDI I 20-/u1/inst_565_ff_d_0_dff/Q2
// "OUT" O 26- 27-
```

back

```
// /u1/inst_565_ff_d_1_13 (14) TIE0
// "OUT" O 269- 268-
```

How to Report on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

forward

The following example shows how to use report_gates and forward to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
// /u1/inst_565_ff_d_1_13 (269) LA
// "S" I 14-
// "R" I 145-
// SCLK I 4-/clk
// D I 265-/u1/_g32/X
// ACLK I 2-/scan_mclk
// SDI I 20-/u1/inst_565_ff_d_0_dff/Q2
// "OUT" O 26- 27-
```

forward

```
// /u1/inst_565_ff_d_1_13 (26) BUF
// "I0" I 269-
// "OUT" O 268- 75-
```

forward

```
// /u1/inst_565_ff_d_1_13 (268) LA
// "S" I 14-
// "R" I 145-
// BCLK I 1-/scan_sclk
// "D0" I 26-
// "OUT" O 24- 25-
```

Flattening Rule Violations

If you encounter flattening rule violations, you can use the report_flattener_rules command to display either a summary of the flattening rule violations or the data for a specific violation, for troubleshooting purposes.

In addition, you can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate rules.

Design Rule Output

Many error messages provide a line number and file name to help you resolve the error condition. However, some error messages may relate to problems with internally generated data and do not provide a line number and file name.

For example, a syntax error (P1) in a test procedure file would produce the following message:

```
The following occurred at line 10 in file testproc
Syntax error in line number 10. (P1-1)
```

Design Rules

Each DRC class has multiple DRCs.

RAM Rules (A Rules)	2666
A1	2666
A2	2667
A3	2668
A4	2668
A5	2669
A6	2670
A7	2670
A9	2671
A10	2672
A11	2672
A12	2673
A13	2674
A14	2675
A15	2675
A16	2676
A17	2676
A18	2677
Assertion Rules (ASSERT Rules).....	2680
Enforcement of Assertions	2680
ASSERT1	2680
ASSERT2	2682
ASSERT3	2683
BIST Rules (B Rules).....	2686
B2	2686
B4	2686
Clock Rules (C Rules)	2687
Clock Terminology	2687
The ATPG Analysis Option	2692
C1	2692
C2	2697
C3	2698
C4	2705
C5	2711
C6	2713
C7	2716
C8	2717
C9	2718
C10	2720
C16	2721
C17	2722

C19	2723
C21	2724
C22	2725
C23	2728
C24	2729
C25	2730
Scan Cell Data Rules (D Rules)	2732
D1	2732
D2	2737
D3	2738
D4	2738
D5	2739
D6	2740
D7	2741
D8	2742
D9	2743
D10	2744
D12	2746
Pre-DFT Clock Rules (DFT_C Rules)	2747
Simulation Contexts for Pre-DFT DRCs	2747
DFT_C1	2751
DFT_C2	2754
DFT_C3	2755
DFT_C4	2756
DFT_C5	2757
DFT_C6	2758
DFT_C7	2760
DFT_C8	2761
DFT_C9	2762
DFT_C10	2765
DFT_C11	2771
DFT_C12	2772
DFT_C13	2773
Extra Rules (E Rules)	2775
E1	2775
E2	2776
E3	2776
E4	2777
E5	2780
E6	2782
E7	2783
E8	2783
E9	2784
E10	2785

E11	2790
E12	2791
E13	2791
E14	2792
E15	2795
EDT Finder Rules (F Rules)	2796
F3	2796
F4	2797
F5	2797
F6	2798
F7	2800
F8	2800
F9	2806
F10	2807
F11	2809
F12	2809
F13	2809
F14	2810
F15	2811
F16	2811
F17	2812
F18	2813
F19	2813
F20	2814
F21	2816
F22	2817
Flattening Rules (FN, FP, and FG Rules)	2821
FN1	2822
FN2	2822
FN3	2822
FN4	2822
FN5	2823
FN6	2823
FN7	2823
FN8	2823
FN9	2824
FP1	2824
FP2	2824
FP3	2824
FP4	2824
FP5	2825
FP6	2825
FP7	2825
FP8	2825
FP9	2826

FP10	2826
FP11	2826
FP12	2826
FP13	2826
FG1	2827
FG2	2827
FG3	2827
FG4	2827
FG5	2828
FG6	2828
FG7	2828
FG8	2828
General Rules (G Rules)	2829
G1	2829
G2	2829
G3	2830
G4	2830
G5	2830
G6	2831
G7	2831
G8	2831
G9	2832
G10	2832
G11	2833
G12	2833
ICL Extraction Rules (I Rules).....	2834
I1	2834
I2	2835
I3	2837
I4	2838
I5	2839
I6	2840
I7	2842
ICL Semantic Rules (ICL Rules)	2843
ICL1	2847
ICL2	2848
ICL3	2848
ICL4	2848
ICL5	2849
ICL6	2849
ICL7	2849
ICL8	2849
ICL9	2850
ICL10	2850

ICL11	2850
ICL12	2851
ICL13	2851
ICL14	2851
ICL15	2851
ICL16	2852
ICL17	2852
ICL18	2852
ICL19	2853
ICL20	2853
ICL21	2853
ICL22	2853
ICL23	2854
ICL24	2854
ICL25	2854
ICL26	2854
ICL27	2855
ICL28	2855
ICL29	2855
ICL30	2855
ICL31	2856
ICL32	2856
ICL33	2856
ICL34	2857
ICL35	2857
ICL36	2857
ICL37	2857
ICL38	2858
ICL39	2858
ICL40	2858
ICL41	2859
ICL42	2859
ICL43	2859
ICL44	2860
ICL45	2860
ICL48	2860
ICL49	2860
ICL50	2861
ICL51	2861
ICL52	2862
ICL53	2862
ICL54	2862
ICL55	2863
ICL56	2863
ICL57	2863

ICL58	2863
ICL59	2864
ICL60	2864
ICL61	2864
ICL62	2865
ICL63	2865
ICL64	2865
ICL65	2865
ICL66	2866
ICL67	2866
ICL68	2866
ICL70	2866
ICL71	2867
ICL75	2870
ICL76	2870
ICL77	2870
ICL78	2871
ICL79	2871
ICL80	2872
ICL81	2872
ICL82	2872
ICL83	2873
ICL84	2873
ICL85	2873
ICL86	2873
ICL87	2874
ICL88	2874
ICL89	2874
ICL90	2875
ICL91	2876
ICL92	2876
ICL93	2876
ICL94	2877
ICL95	2877
ICL96	2877
ICL97	2877
ICL98	2878
ICL99	2878
ICL100	2878
ICL101	2879
ICL102	2879
ICL103	2879
ICL104	2880
ICL105	2880
ICL106	2881

ICL107	2881
ICL108	2882
ICL109	2882
ICL110	2882
ICL111	2882
ICL112	2883
ICL113	2883
ICL114	2884
ICL115	2884
ICL116	2884
ICL117	2885
ICL118	2885
ICL119	2885
ICL120	2886
ICL121	2886
ICL122	2886
ICL123	2886
ICL124	2887
ICL125	2887
ICL126	2888
ICL127	2888
ICL128	2888
ICL129	2889
ICL130	2889
ICL131	2890
ICL132	2890
ICL133	2891
ICL134	2891
ICL135	2892
ICL136	2893
ICL137	2893
EDT Rules (K Rules)	2895
K1	2896
K2	2896
K3	2896
K4	2897
K5	2897
K6	2897
K7	2899
K8	2899
K9	2899
K10	2901
K11	2901
K12	2902
K13	2902

K14	2902
K15	2903
K16	2908
K17	2908
K18	2909
K19	2909
K20	2910
K21	2911
K22	2911
K23	2913
K24	2913
K25	2914
Procedure Rules (P Rules)	2916
P1	2918
P2	2919
P3	2919
P4	2920
P5	2920
P6	2920
P7	2921
P8	2921
P9	2921
P10	2922
P11	2922
P12	2923
P13	2924
P14	2924
P15	2925
P16	2925
P17	2925
P18	2926
P19	2926
P20	2927
P21	2927
P22	2927
P23	2928
P24	2928
P25	2929
P26	2929
P27	2929
P28	2930
P29	2931
P30	2931
P31	2932
P32	2932

P33	2932
P34	2933
P35	2933
P36	2934
P37	2934
P38	2934
P39	2935
P40	2935
P41	2935
P42	2936
P43	2936
P44	2937
P45	2937
P46	2938
P47	2938
P48	2938
P49	2939
P50	2939
P51	2939
P52	2940
P53	2940
P54	2941
P55	2942
P56	2942
P58	2943
P59	2943
P60	2943
P62	2944
P63	2944
P64	2945
P65	2945
P66	2945
P70	2946
P71	2946
P72	2947
P73	2947
P74	2947
P75	2948
P76	2949
P77	2949
P78	2950
P79	2950
P80	2951
P81	2952
P82	2952

P83	2953
P84	2953
P85	2954
P86	2954
P87	2955
P88	2955
P89	2956
P90	2956
P91	2957
P92	2957
P93	2958
P94	2958
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)	2961
R1	2961
R2	2962
R3	2962
R4	2963
R5	2963
R6	2964
R7	2964
R8	2965
R9	2968
R10	2969
R11	2970
R12	2971
R13	2972
R14	2972
R15	2974
R16	2977
R17	2979
R18	2979
R19	2980
R20	2980
R22	2981
R23	2982
R27	2982
R28	2983
Scannability Rules (S Rules)	2984
S1	2985
S2	2986
S3	2987
S4	2988
S5	2989
S6	2990
S7	2991

S8	2992
Scan Chain Trace Rules (T Rules)	2994
T1	2995
T2	2995
T3	2996
T4	3000
T5	3005
T6	3007
T8	3008
T9	3008
T10	3009
T11	3009
T12	3009
T13	3010
T14	3010
T15	3010
T16	3011
T17	3011
T18	3012
T19	3012
T20	3013
T21	3013
T22	3013
T23	3014
T24	3014
T25	3015
T26	3016
Power-Aware Rules (V Rules)	3018
V1	3019
V2	3020
V3	3020
V4	3020
V5	3021
V6	3021
V7	3022
V8	3022
V9	3023
V10	3024
V11	3025
V12	3025
V13	3026
V14	3027
V15	3028
V16	3028
V17	3028

V18	3029
V19	3029
V20	3030
V21	3030
Timing Rules (W Rules)	3032
W1	3033
W2	3033
W3	3034
W4	3034
W5	3034
W6	3035
W7	3035
W8	3036
W9	3036
W10	3036
W11	3037
W12	3037
W13	3038
W14	3038
W15	3038
W17	3039
W18	3039
W19	3040
W20	3040
W21	3041
W22	3041
W23	3041
W24	3042
W25	3042
W26	3043
W27	3043
W28	3044
W29	3044
W30	3045
W31	3045
W32	3046
W33	3046
W34	3047
W35	3048
W36	3048
W37	3048
W38	3049
W39	3050
W41	3051
Other DRC Messages.	3052

Transparent Capture Handling Analysis	3052
Oscillation Limitation	3053
RAM Summary Results and Test Capability	3053

RAM Rules (A Rules)

The tool checks RAM gates to identify proper test methods. You can select the handling of any RAM rule to be error, warning, note, or ignore.

Tip

 In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

A1.....	2666
A2.....	2667
A3.....	2668
A4.....	2668
A5.....	2669
A6.....	2670
A7.....	2670
A9.....	2671
A10.....	2672
A11.....	2672
A12.....	2673
A13.....	2674
A14.....	2675
A15.....	2675
A16.....	2676
A17.....	2676
A18.....	2677

A1

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When all write control lines are at their off-state, all write, set, and reset inputs of RAMs must be at their inactive state. The tool performs this check using the simulated values that result when all defined write control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any write, set, or reset input of any RAM gate is not off.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed on the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write control lines (including those that are RAM set and reset lines) or defining the wrong off-state.

The occurrence message is:

```
Write controls off failed to force off RAM T line of N (G). (A1-1)
```

T is the type of input (write, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, and A1-1 is the rule and violation ID number.

The summary message is:

```
N RAM write/set/reset lines not forced off when write controls are off.  
(A1)
```

N is the number of occurrences of rules violation A1.

A2

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

A defined scan clock must not propagate to a RAM gate, except for its read lines. The tool performs this check by determining the forward cone of influence for all clock pins (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when a RAM gate is in a clock cone.

The default setting for this DRC is “Ignore.” You can enable this check with the set_drc_handling command. When the DRC is enabled and an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Scan clock is connected to RAM N (G). (A2-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

N RAMs are connected to a scan clock. (A2)

N is the number of occurrences of rules violation A2.

A3

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A write or read control line must not propagate to an address line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an address line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

RAM control line connected to address input of RAM N (G) . (A3-1)

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

N RAM address inputs are connected to a RAM control line. (A3)

N is the number of occurrences of rules violation A3.

A4

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A write or read control line must not propagate to a data line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained value, and the initialized non-scan cells are set to their stable stated. The rule violation occurs when a data-in line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
RAM control line connected to data input of RAM N (G) . (A4-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM data inputs are connected to a RAM control line. (A4)
```

N is the number of occurrences of rules violation A4.

A5

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A RAM gate must not propagate to another RAM gate. The tool performs this check by determining the forward cone of influence for all RAM gates (RAM cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an input of a RAM gate is in the RAM cone.

The default handling for this rule violation is warning. The occurrence message is:

```
RAM N1 (G1) connected to RAM N2 (G2) . (A5-1)
```

N1 is the instance name of one RAM gate, G1 is its gate ID number, N2 is the instance name of the other RAM gate, and G2 is its gate ID number.

The summary message is:

N RAMs are connected to RAMs. (A5)

N is the number of occurrences of rules violation A5.

A6

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All the write inputs of all RAMs and all the read inputs of all data_hold RAMs must be at their off-state during all test procedures, except **test_setup**. The tool performs this check using the simulated values that result when it simulates the test procedures. The rule violation occurs if any write, set, or reset input of any RAM, or a read input of a data_hold RAM is not off.

The default handling for this rules violation is warning. Failure to satisfy this rule for write inputs results in the RAM being unavailable to hold its contents during scan operation, which may cause a loss in test coverage. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write or read control lines, or defining the wrong off-state.

The occurrence message is:

L line of RAM N (G) not off during time T of P procedure. (A6-1)

L is the type of input (write, read, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, T is the time, and P is the procedure name.

The summary message is:

There were N occurrences of uncontrolled RAMs during test procedures.
(A6)

N is the number of occurrences of rules violation A6.

A7

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When all read control lines are at their off-state, all read inputs of RAMs with the `read_off` attribute set to hold must be at their inactive state. The tool performs this check using the simulated values that result when all defined read control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any read input of a `data_hold` RAM gate is not off.

The default handling for this rules violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to `error_pattern` and using the `report_gates` command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all read control lines or defining the wrong off-state.

The occurrence message is:

```
Read controls off failed to force off RAM read line of N (G). (A7-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM read lines not forced off when read controls are off. (A7)
```

N is the number of occurrences of rules violation A7.

A9

Category: RAM

Contexts Supported: `dft -scan`, `patterns -scan_diagnosis`

Default Handling: Warning

`report_drc_rules`: Supported

A read-only RAM model should define an initialization file. If an initialization file is not defined, the tool will treat instances of the RAM as TIEX, reducing test coverage.

The default handling for this rule violation is warning. To correct the problem, be sure the Tessent Cell library model of the RAM includes an `init_file` attribute that correctly specifies an initialization file.

The occurrence message is:

```
The initialization file is not defined for read-only RAM N (G). (A9-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N read-only RAMs do not have initialization file. (A9)
```

N is the number of occurrences of rules violation A9.

A10

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that does not have a write port must not be disturbed by any procedures except test_setup.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during simulation of a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not forcing the primary inputs, or not initializing the state elements, that control the RAM's set or reset port to an appropriate value.

When the A10 and/or A11 rule is violated, the tool will treat the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

```
The content of read-only RAM N (G) can be disturbed through L line during time T of P procedure. (A10-1)
```

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (set or reset), T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of disturbed read-only RAMs during test procedures. (A10)
```

N is the number of occurrences of rules violation A10.

A11

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning
report_drc_rules: Supported

The content of a read-only RAM that does not have a write port must not be disturbed through the set or reset lines during capture.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not constraining the primary inputs or the state elements that control the RAM's set or reset port.

When the A10 and/or A11 rule is violated, the tool will treat the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

The content of read-only RAM N (G) can be disturbed through L line in capture window. (A11-1)

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (set or reset).

The summary message is:

There were N occurrences of disturbed read-only RAMs in capture window. (A11)

N is the number of occurrences of rules violation A11.

A12

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that has a write port must not be disturbed by any procedure except test_setup (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

When rule A12 and/or A13 is violated, the tool will ignore the RAM's contents initialized through the initialization file.

The occurrence message is:

The initialization file of RAM N (G) cannot be used during test pattern generation since RAM content can be disturbed through L line during time T of P procedure. (A12-1)

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (write, set or reset), T is the time, and P is the procedure name.

The summary message is:

There were N occurrences of disturbed RAM initialization file during test procedures. (A12)

N is the number of occurrences of rules violation A12.

A13

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that has a write port must not be disturbed through the write, set or reset lines during capture (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

When rule A12 and/or A13 is violated, the tool will ignore the RAM's contents initialized through the initialization file.

The occurrence message is:

The initialization file of RAM N (G) cannot be used during test pattern generation since RAM content can be disturbed through L line in capture window. (A13-1)

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (write, set or reset).

The summary message is:

```
There were N occurrences of disturbed RAM initialization file in capture  
window. (A13)
```

N is the number of occurrences of rules violation A13.

A14

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be observable. The tool performs this check by forcing all pin constraints and tied cells and check if there is any sensitizable path from the outputs of a RAM to any observation point. The rule violation occurs if there is none.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) is unobservable. (A14-1)
```

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints blocking the output of the RAM.

A15

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to read. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the read enable and read clock lines of each read control port of a RAM. The rule violation occurs if the simulation values indicate that all read control ports are off.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot read because its read control ports are  
// constrained off. (A15-1).
```

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints that prevent the read enable or read clock lines of a RAM to be turned ON.

A16

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to write. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the write enable and write clock lines of each write control port of a RAM. The rule violation occurs if the simulation values indicate that all write control ports are off. This rule checking will not be performed for read-only ROMs.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot write because its write control ports are
// constrained off. (A16-1).
```

N is the instance name of the RAM gate and G is its gate ID number.

A17

Category: RAM

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a RAM cannot capture fault effects because its read port cannot be set to an active state.

The information provided by A17 helps you debug low fault coverage caused by read port issues. To check for A17, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port. If the constrained value of the RAM's read_enable or read_clock is X, and if either of two signals cannot be justified to 1, the tool reports an A17 violation.

To minimize the number of violations reported, the tool traces from rule violation read ports to a common source point, which can be a primary input or internal pin. The violating sources are

reported in order by number of affected elements with the first violation having the most, and so on. This is an example of the reported message when the read enable or read clock is X:

report_drc_rule A17

```
// Warning: Internal gate N1 (G1) has a value of X and cannot be justified
// to 1.
// This prevents read by C memory elements, one of which is N2 (G2).
// (A17-1)
```

N1 is the name of the gate with X value, G1 is the gate ID of that gate. C is the number of memory elements, N2 is the name of the gate whose read is prevented, G2 is the gate ID of that gate.

When a single primary input (PI) drives both the read_enable and the read_clock which are inverted from each other, this is the warning message:

```
// Warning: Primary input P (S) cannot be justified to both 0 and 1
// simultaneously to control read enable and clock ports that are
// inverted from each other.
// This prevents read by C memory elements, one of which is N (G).
// (A17-1)
```

P is the primary input name, S is the net ID number, C is the count of memory elements, N is the gate name, and G is the gate ID number.

When the read clock and read enable pins cannot be justified to the on state simultaneously, this is the warning message:

```
// Warning: Internal gate N1 (G1) and N2 (G2) that control read enable
// and clock respectively cannot be justified to 1 and 1 simultaneously.
// This prevents read by C memory elements, one of which is N3 (G3).
// (A17-1)
```

N1 is the name of the read_enable gate, G1 is the gate ID of that gate. N2 is the name of the read_clock gate, G2 is the gate ID number of that gate, C is the number of memory elements, N3 is the name of the gate whose read is prevented, G3 is the gate ID of that gate.

The default handling of A17 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
A17: #fails=1 handling=warning (memory elements cannot read because clock
ports cannot be fully controlled).
```

To debug A17 violations, use the “[set_gate_report -constrain_value on](#)” command to look at the constrained values on the read ports of affected RAMs.

A18

Category: RAM

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a RAM cannot capture fault effects because its write port cannot be set to an active state.

The information provided by A18 helps you debug low fault coverage caused by write port issues. To check for A18, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port. If the constrained value of the RAM's write_enable or the write_clock is X, and if either of the two signals cannot be justified to 1, the tool reports an A18 violation.

To minimize the number of violations reported, the tool traces from rule violation write ports to a common source point, which can be a primary input or internal pin. The violating sources are reported in order by number of affected elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule A18

```
// Warning: Internal gate N1 (G1) has a value of X and cannot be justified
to 1.
//           This prevents write by C memory elements, one of which is N2
(G2). (A18-1)
```

N1 is the name of the gate with X value, G1 is the gate ID of that gate. C is the number of memory elements, N2 is the name of the gate whose write is prevented, G2 is the gate ID of that gate.

When a single primary input (PI) drives both the write_enable and the write_clock which are inverted from each other, this is the warning message:

```
// Warning: Primary input P (S) cannot be justified to both 0 and 1
simultaneously to control write enable and clock ports that are inverted
from each other.
//           This prevents write by C memory elements, one of which is N
(G). (A18-1)
```

P is the primary input name, S is the net ID number, C is the count of memory elements, N is the name of the gate whose write is prevented, and G is the gate ID number of that gate.

When the write clock and write enable pins cannot be justified to the on state simultaneously, this is the warning message:

```
// Warning: Internal gate N1 (G1) and N2 (G2) that control write enable
and clock respectively cannot be justified to 1 and 1 simultaneously.
//           This prevents write by C memory elements, one of which is N3
(G3). (A18-1)
```

N1 is the name of the write_enable gate, G1 is the gate ID of that gate. N2 is the name of the write_clock gate, G2 is the gate ID number of that gate, C is the number of memory elements, N3 is the name of the gate whose write is prevented, G3 is the gate ID of that gate.

The default handling of A18 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
// Warning: There were 264 A18 violations (memory elements cannot write because clock ports cannot be fully controlled).
```

To debug A18 violations, use the “[set_gate_report](#)-constrain_value on” command to look at the constrained values on the write ports of affected RAMs.

Assertion Rules (ASSERT Rules)

DRC checks that the assertion values on the pins of models from read_cell_library occur during DRC simulation over the specified period.

Enforcement of Assertions	2680
ASSERT1	2680
ASSERT2	2682
ASSERT3	2683

Enforcement of Assertions

DRC checks that instances of read_cell_library models with an assert_shift, assert_capture or assert_pattern have the value specified at the time the assertion is enforced.

Currently, there are a total of five phases to a typical ATPG simulation, as follows:

1. test_setup before 1st load_unload.
2. load_unload before shift.
3. shift procedures (and the changes between them).
4. Going from shift to capture at end load_unload when pin constraints are forced.
5. Capture.

assert_shift is enforced in phase 3 only (no assertion checks in 1,2,4,5).

assert_capture is enforced in phase 5 only (no assertion checks in 1,2,3,4).

assert_pattern is enforced in phases 2,3,4,5 (no assertion checks in 1).

ASSERT1

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Values defined on Tessent cell library model pins via assert_shift should be as defined.

Values which should occur on all instances of a model throughout shift should be specified by an assert_shift on the input of the model. These values can be specified in Tessent cell library

syntax or as Verilog assertions using Verilog syntax. Libcomp can be used to translate the Verilog assertions into Tessent syntax. For example:

```
module mux_scan (Q, CK, D, SI, SE);
    output Q;
    reg Q;
    input CK;
    input D;
    input SI;
    (* assert_shift = 1*) input SE;
    assign Din = SE ? SI : D;
    always @ (posedge CK) begin
        Q <= Din;
    endmodule
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tessent syntax.

```
model mux_scan
    (Q, CK, D, SI, SE)
(
    model_source = verilog_module;
    input (CK) ( )
    input (D) ( )
    input (SI) ( )
    input (SE) ( assert_shift = 1)
    output (Q) ()
)
(
    primitive = _mux  mlc_mux_1 ( D, SI, SE, Din );
    primitive = _dff  mlc_dff_1 ( , , CK, Din, Q, );
)
)
```

Netlist instances with assert_shift are checked against stable shift simulation values for compliance. Violations default to error which halts the tool and returns to the prompt. set_drc_handling can be used to change the default to Warning, Note or Ignore.

Example:

```
// command: set_drc_handling assert1 warn verbose
```

An example violation after the above setting follows:

```
// Warning: assert_shift=Stable but sim val = "X" (not known and
// constant) from driver '/clk_01' (11)
// Assertion on instance/pin 'ff10/CLK'
// Assertion came from pin 'CLK' of model 'sff'
//           line 500 file 'data/adk.atpg' (ASSERT1-1)
```

report_drc_rules supports assert_shift (ASSERT1) as follows:

```
// command: report drc rules assert1
// Warning: assert_shift=Stable but sim val = "X" (not known and
// constant) from driver '/clk_01' (11)
//      Assertion on instance/pin 'ff10/CLK'
//      Assertion came from pin 'CLK' of model 'sff'
//          line 500 file 'data/adk.atpg' (ASSERT1-1)
```

report_gate also supports ASSERT1 (assert_shift) after issuing:

```
set_gate_report drc_pattern stable_shift
```

You can have an assert_shift and an assert_capture both on the same input, as shown below:(*)

```
assert_shift = 1, assert_capture = 0 *) input SE; // Verilog syntax
input (SE) (assert_shift = 1; assert_capture = 0) // Tesson syntax
```

ASSERT2

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Values defined on Tesson cell library model pins via assert_capture should be as defined.

Values which should occur on all instances of a model throughout capture should be specified by an assert_capture on the input of the model. These values can be specified in Tesson cell library syntax or as Verilog assertions using Verilog syntax. Libcomp can be used to translate the Verilog assertions into Tesson syntax. For example:

```
module mux_scan (Q, CK, D, SI, SE);
    output Q;
    reg Q;
    input CK;
    input D;
    input SI;
    (*assert_capture = 0*) input SE;
    assign Din = SE ? SI : D;
    always @ (posedge CK) begin
        Q <= Din;
    endmodule
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tesson syntax.

```

model mux_scan
  (Q, CK, D, SI, SE)
(
  model_source = verilog_module;
  input (CK) ( )
  input (D) ( )
  input (SI) ( )
  input (SE) ( assert_capture = 0 )
  output (Q) ( )
(
  primitive = _mux  mlc_mux_1 ( D, SI, SE, Din );
  primitive = _dff  mlc_dff_1 ( , , CK, Din, Q, );
)
)

```

Netlist instances with assert_capture are checked against stable capture simulation values for compliance. Violations default to error which halts the tool and returns to the prompt. set_drc_handling can be used to change the default to Warning, Note or Ignore.

Example:

```
// command: set_drc_handling assert2 note verbose
```

An example violation after the above setting follows:

```

// Note: assert_capture=0 but sim val = "X" (not known and constant) from
driver '/d1' (1)
// Assertion on instance/pin 'ff10/D'
// Assertion came from pin 'D' of model 'sff'
// line 500 file 'data/adk.atpg' (ASSERT2-1)

```

report_drc_rules supports assert_capture (ASSERT2) as follows:

```

// command: report drc rules assert2
// Note: assert_capture=0 but sim val = "X" (not known and constant) from
driver '/d1' (1)
// Assertion on instance/pin 'ff10/D'
// Assertion came from pin 'D' of model 'sff'
// line 500 file 'data/adk.atpg' (ASSERT2-1)

```

report_gate also supports ASSERT2 (assert_capture) after issuing:
set_gate_report drc_pattern stable_capture

You can have an assert_shift and an assert_capture both on the same input, as shown below:(*
assert_shift = 1, assert_capture = 0 *) input SE; // Verilog syntax
input (SE) (assert_shift = 1; assert_capture = 0) // Tesson syntax

ASSERT3

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Values defined on Tessent cell library model pins via assert_pattern should be as defined.

Values which should occur on all instances of a model throughout the entire pattern should be specified by an assert_pattern on the input of the model. These values can be specified in Tessent cell library syntax or as Verilog assertions using Verilog syntax. Libcomp can be used to translate the Verilog assertions into Tessent syntax. For example

```
primitive prim_not(out, in);
(* Assert_capture = "1'b0", ASSERT_pattern = "stable" *)
  input in;
(* assert_capture = "Stable", assert_pattern = "Z", assert_shift = 1'bz *)
  output out;
  table
    // in : out
    0 : 1 ;
    1 : 0 ;
    X : X ;
  endtable
endprimitive
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tessent syntax.

```
model prim_not
  (out, in)
(
  model_source = verilog_udp;

  input (in)  (  Assert_capture = 1'b0;    ASSERT_pattern = stable;  )
  output (out) (  assert_capture = Stable;   assert_pattern = Z;   assert_shift = 1'bz;
)  (
  primitive = _inv mlc_gate0 (in, out);
)
)
```

Netlist instances with assert_pattern are checked against stable pattern simulation values for compliance. Violations default to error which halts the tool and returns to the prompt. set_drc_handling can be used to change the default to Warning, Note or Ignore.

Example:

```
// command: set_drc_handling assert3 warn verbose
```

Two example violations after the above setting follow. Note that the second example says to see stable_shift, because the pattern violation occurred within shift. This is more helpful for finding the root cause of the violation to fix it. If the violation is only in capture, it will indicate that. The first violation shown below does not occur within shift or within capture, so it says to see stable_after_setup which shows values across the entire pattern.

```
// Warning: assert_pattern=1 but sim val = "X" (not known and constant)
(see stable_after_setup) from driver '/don_in' (10)
// Assertion on instance/pin 'not1/A'
// Assertion came from pin 'A' of model 'inv01'
// line 266 file 'data/adk.atpg' (ASSERT3-1)
// Warning: assert_pattern=1 but sim val = "0" (see stable_shift) from
driver '/not1/Y' (64)
// Assertion on instance/pin 'not2/A'
// Assertion came from pin 'A' of model 'inv01'
// line 266 file 'data/adk.atpg' (ASSERT3-2)
```

report_drc_rules supports assert_pattern (ASSERT3) as follows:

```
// command: report drc rules assert3
// Warning: assert_pattern=1 but sim val = "X" (not known and constant)
(see stable_after_setup) from driver '/don_in' (10)
// Assertion on instance/pin 'not1/A'
// Assertion came from pin 'A' of model 'inv01'
// line 266 file 'data/adk.atpg' (ASSERT3-1)
```

report_gate also supports ASSERT3 (assert_pattern) after issuing:

set_gate_report drc_pattern stable_after_setup

However, as stated above, it may be more helpful to use stable_shift or stable_capture, and the violation message will indicate when this is true.

BIST Rules (B Rules)

Whenever LFSRs are defined, the tool performs BIST rules checking to ensure proper application of BIST patterns to the circuit. You cannot change the handling of the BIST rules from their default conditions of either error or warning—with the exception of rule B2.

B2	2686
B4	2686

B2

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Every scan chain input pin must connect to an LFSR. Correct this error condition by connecting the scan chain input pin of the indicated chain to an LFSR.

You can change how the rules checker handles this check with the set_drc_handling command.

The error message is:

```
Input of chain C has no LFSR connection. (B2-1)
```

C is the name of the scan chain.

B4

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Every scan chain output pin must connect to an LFSR. Correct this error condition by connecting the scan chain output pin of the indicated chain to an LFSR.

The error message is:

```
Output of chain C has no MISR connection. (B4)
```

C is the name of the scan chain.

Clock Rules (C Rules)

The application checks the scan clocks to ensure their proper definition and operation. You may select the handling of any clock rule to be error, warning, note, or ignore.

The following subsections describe the clock rules and the special handling you can set for them.

Tip

 In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

Clock Terminology	2687
The ATPG Analysis Option	2692
C1	2692
C2	2697
C3	2698
C4	2705
C5	2711
C6	2713
C7	2716
C8	2717
C9	2718
C10	2720
C16	2721
C17	2722
C19	2723
C21	2724
C22	2725
C23	2728
C24	2729
C25	2730

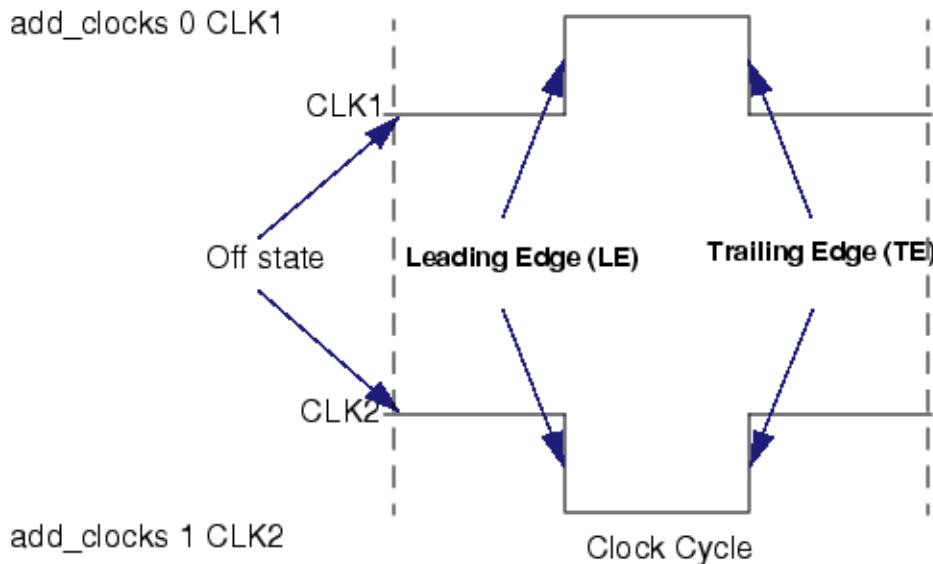
Clock Terminology

The clock rule information contains a couple of recurring concepts. To make optimal use of the information, you should understand these concepts.

Clock Signals

The tool considers any signal to be a clock if it can change the state of a sequential element, including system clocks, sets, and resets. When you define each clock primary input (a required setup step you perform using the `add_clock` command or “`analyze_control_signals -Auto_fix`”), a key piece of information specified with either of these commands is the pin value that represents the clock’s off state. Two important terms arise out of this definition, as shown in the following figure.

Figure 7-4. Clock Cycle Terminology



The transition of the clock from the off state to the on state is considered the leading edge of the clock, while the transition from the on state to the off state is considered the trailing edge of the clock.

Clock Cones and Effect Cones

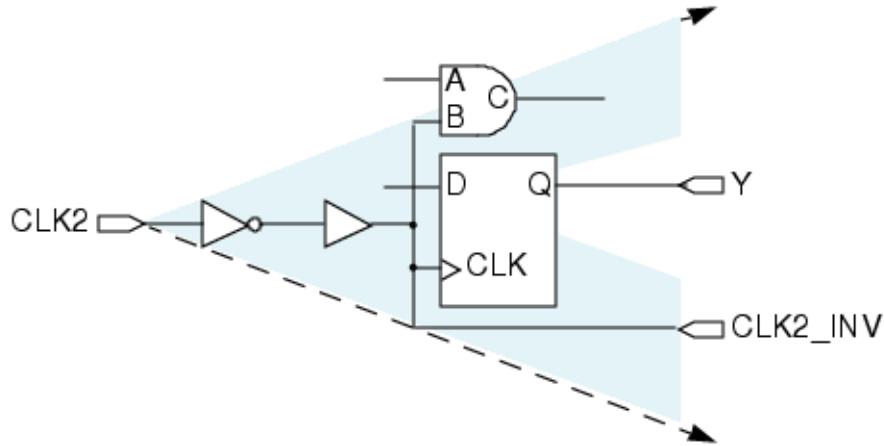
A gate pin or output pin is considered to be in the *clock cone* of a clock signal when they are connected through combinational logic gates and transparent latches (TLAs) to the clock primary input.

Note

 The tools treat a latch as a transparent latch (TLA) rather than as a normal level-sensitive sequential element (LA) if it will be continuously enabled at the start of the capture cycle when all clocks are at their defined off values. This treatment therefore depends on tool setups and how the latch is wired into the design. A TLA passes values without holding state and thus acts like a buffer. For more information, refer to “[Simulation Primitives of the Flattened Model](#)” in the *Tessent Scan and ATPG User’s Manual*.

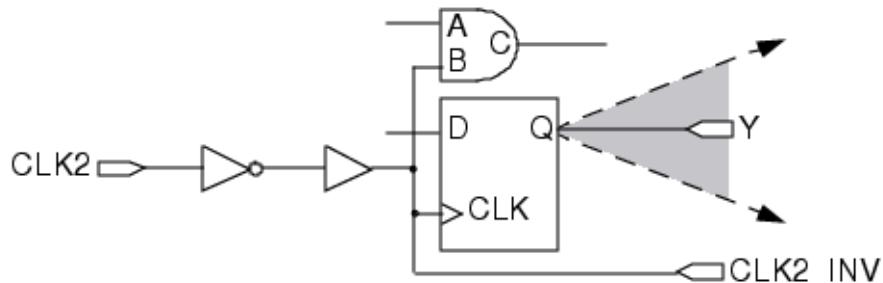
The clock cone is basically the fanout of the clock signal through strictly combinational logic and TLAs. Pins B, C, CLK and CLK2_INV in the circuit excerpt shown in the following figure are all in CLK2's clock cone.

Figure 7-5. Example Clock Cone



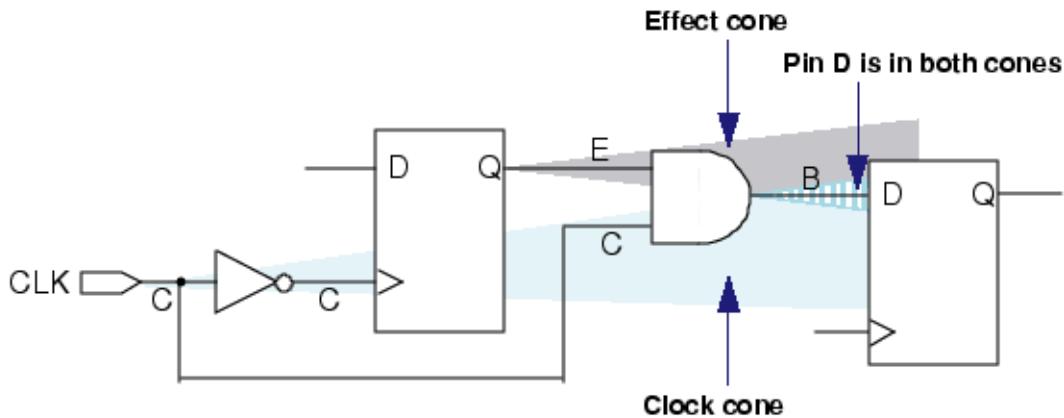
The tools consider a gate pin or output pin to be in a clock's *effect cone* if there is a sequential element between the clock net and the gate pin or output pin. In the following figure, pin Y is in the effect cone of CLK2.

Figure 7-6. Example Effect Cone



A pin can be in both the *clock cone* and the *effect cone* as shown for pin D on one of the flip-flops in the following figure.

Figure 7-7. Pin in Both Cones

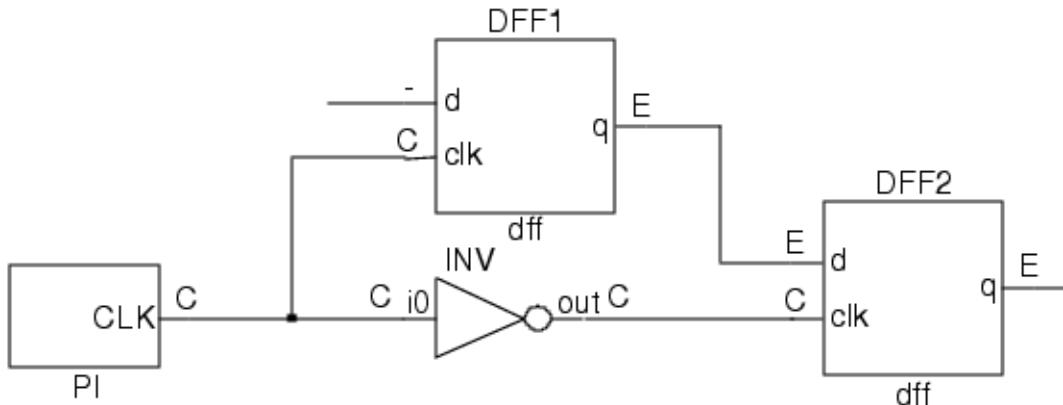


You can get the Flat Schematic window of DFTVisualizer to show a C, E or B near device pins that are in the clock (C), effect (E), or both (B) cones of a particular clock by issuing the `set_gate_report` command with the `Clock_cone` option after the tool has flattened the design. For example:

```
create_flat_model  
set_gate_report clock_cone CLK
```

Alternatively, you can choose **Data > Clock Cone** from the DFTVisualizer menu (with the Flat Schematic window active) and in the dialog box specify the clock. The following figure shows an example of how this might look for a circuit fragment in the Flat Schematic window of DFTVisualizer.

Figure 7-8. DFTVisualizer Flat Schematic Window View Showing Clock Cones



When the `Clock_cone` option is in effect, the output of the `report_gates` command also displays clock cone data:

report_gates /DFF1

```
// /DFF1 dff
//   clkI  (C) 2-/CLK
//   d   I  (-) 1-/C
//   q   O  (E) 5-
```

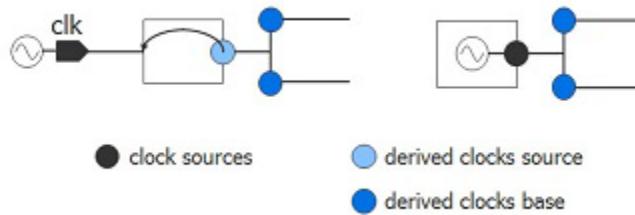
Note

 The tool shows clock and effect cone data for just one clock at a time—the clock whose primary input *pin_name* you specified in the last “set_gate_report Clock_cone” command.

Clock Source and Derived Clocks

For clock rule checks, Scan Cell Data Rules (D rules) through DFT_C5 (Memory BIST clock tracing), three different types of clocks can be defined at primary ports or instance pins: synchronous and asynchronous clock sources, derived clock sources, and derived clock branches. The differences between these clock types are shown in the following figure:

Figure 7-9. Clock Source And Derived Clocks



The different clock types are described as follows:

- **clock source (synchronous or asynchronous)** — is defined at a primary or instance pin where a clock pulse is available in the design. It can be a well-defined pin where an external clock generator is connected, or it can be a well-defined instance pin where an internal clock generator produces a clock signal.
- **derived clock source** — is a defined instance pin where a reference pin has to be specified because a tracing path to the corresponding “clock source” is blocked. One typical case is a PLL instance represented as a blackbox in the design. Using the add_clocks -reference option, you can link the derived clock source instance pin to a clock source.
- **derived clock branch** — a derived clock branch describes the case in which the clock tree of a design, sometimes different branches of the tree, can be unbalanced even if they are derived from the same clock source. That is, clock branches are not guaranteed to be synchronous between these clock branches. You define such a clock branch using the add_clocks -branch option. A derived clock branch must always be driven by a “clock source” or a “derived clock source”. Clock branches can also be used for later network manipulations.

The ATPG Analysis Option

Clock rules C1, C3, C4, and C5 can run full ATPG analysis during their checks.

For more information on ATPG analysis, refer to “[How to Turn on ATPG Analysis](#).”

The Memory BIST Analysis Option

Clock rule checks DFT_C1 through DFT_C5 run only in the dft context, and only if there are memory instances in the design and the `set_dft_specification_requirements-memory_test` option is set to “on”.

These clock rule checks are skipped entirely for all other cases. For more information, see the [set_dft_specification_requirements](#) command.

C1

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A scan or non-scan cell must not capture data when all specified clocks are set to their off states.

When all clocks are at their off state as defined with the `add_clocks` command, all clock inputs (including sets and resets) of scan and non-scan cells must be at their off state. For non-scan cell violations, the tool converts these to TIEX.

In the case of a DFF scan cell, the clock input must be a stable binary value (0 or 1).

Note

 If the clock input value of a DFF scan cell simulates as X, the tool by default performs additional analysis to determine if the X represents a stable binary value. An X that represents a stable binary value satisfies the C1 rule’s clock input requirement for a DFF. (To turn off the additional analysis, use the `set_drc_handling` command’s `Noatpg_analysis` argument.)

The tool performs this check using the simulated values that result when all defined clocks are at their defined off states, constrained pins are set to their constrained values, initialized non-scan cells (initialized during `test_setup` or `load_unload`) are set to their stable states, and C0, C0DX, C1, and C1DX cell constraint values are applied. The rule violation occurs if any clock input, including set and reset lines, of any scan cell memory element is not at its off state.

When the C1 violation occurs on a set/reset port or a level-sensitive latch port, the tool does not issue an error message and does not stop the DRC process. Instead, the tool issues the following summary messages to indicate such situation exist:

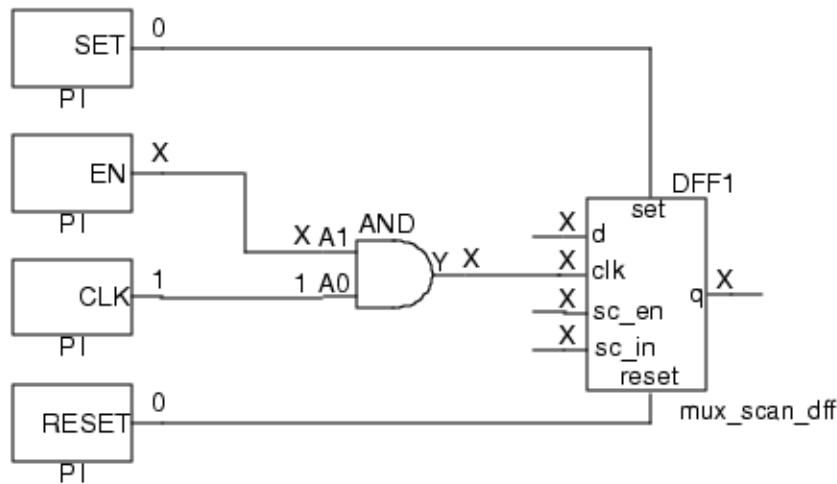
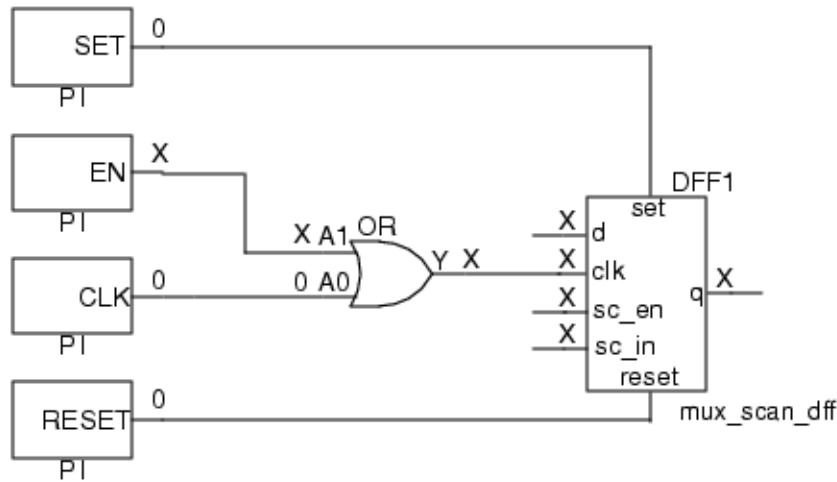
```
// Note: n scan cells whose clock ports are not off when all clocks are
// off will be handled through additional ATPG effort.

// The scan cells have been excluded from the list of C1 DRC violations.
// Use "report_drc_rules C1 -excluded" to report these C1 violations
```

If the number of such scan cells is large, ATPG runtime is increased. One common root cause of these C1 violations is a set/reset pin not defined as a clock. You can define a set/reset pin as a clock to fix the problem and avoid ATPG performance impact.

[Figure 7-10](#) shows an example of a circuit segment that produces a C1 violation due to an indeterminate value (X) at the clk input of DFF1 when the off state of the clocks at the PIs is defined as a logic 0 and all defined clocks are off.

Figure 7-10. C1 Violation Example



The clock input value to the scan cell is X because the EN signal value is X. If EN is left at X, the signal arriving at the clk input of the scan cell cannot be held off.

Effect on Testability

Failure to satisfy this rule can result in unstable scan and non-scan cell values going into or coming out of load_unload, or between cycles. This can result in lower test coverage.

How to Debug C1 Violations

The occurrence message is:

```
// Clock PIs off failed to force off a clock line of T N (G) . (C1-1) .
```

T is the type of scan cell (MASTER, SLAVE, etc.), N is the instance name of the gate, and G is the gate ID number.

The summary message is:

```
// There were N clock rule C1 fails (unstable scan cells when clocks off).
```

N is the number of occurrences of rules violation C1.

For debugging, be sure C1 handling is set to error (the invocation default setting) before you enter a “set_system_mode analysis” command to leave setup mode. Use the following command to check the setting:

```
report_drc_rules -fails_summary
```

If the handling is other than error, you can change it back to error with:

```
set_drc_handling c1 error
```

When C1 handling is set to error, the first C1 violation that occurs will stop the DRC process and return the tool to Setup mode.

Tip

The C1 violation on which the tool stops is the C1 you must debug. Fixing the C1 violations the tool singles out in this manner will often fix other C1's with no extra effort on your part, so is very efficient.

The tool will transcript details about the violation and save the simulation data related to the violation, so you can report it.

You can then debug the violation using DFTVisualizer or by issuing commands from the tool's command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C1 DRC error using DFTVisualizer, use the following command:

```
open_visualizer
```

Choose the **DRC Violations** tab in the Design Browser window, then open the listed DRC's by selecting the “+” symbols as needed to find the violation ID of the desired DRC violation. Double clicking the listed violation, or right clicking and selecting **Analyze DRC Violation** from the popup menu will analyze the violation. The scan cell affected by that occurrence is displayed in the Flat Schematic window.

Tip

i Alternatively, you can issue the analyze_drcViolation command with the C1-1 argument at the tool command line. For example:

analyze_drcViolation c1-1

Once the offending scan cell is displayed, trace backward from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the add_clocks command, is correct. To change the defined off state, delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

How to Debug from the Tool Command Line

To view the location of a C1 DRC error from the tool command line, use the following example command steps:

1. set_drc_handling C1 Error
2. set_gate_report Error_pattern
3. set_system_mode analysis
4. report_drc_rules C1-
5. report_gates *offending_gate's_id#*
6. report_gates -Endpoints -Backward *id#_of_gate_connected_to_offending_gate's_clock*

The following transcript excerpt shows an example of the use of this command sequence starting at step 4:

```
report_drc_rules c1-1
// Error: Clock PIs off failed to force off a clock line of /U$1/DFF1
// (14). (C1-1)
report_gates 14
// /U$1/DFF1 (14) DFF
// "S" I (0) 10-
// "R" I (0) 11-
// clk I (X) 12-/OR/Y
// d I (X) 13-/MUX/out
// "OUT" O (X) 8- 9-
report_gates -endpoints -backward 12
// -----
// Begin backward trace for gate /OR (12).
// -----
// /CLK (3) PI
// CLK O (0) 12-/OR/A0
// /EN (4) PI
// EN O (X) 12-/OR/A1
// Number gates in trace = 2.
```

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. If the off state is correct, the problem typically is due to an indeterminate value (X) in related logic that causes the clock's value at a scan cell to be X. This is the case in the example shown in the figure above. You can fix these by using the add_input_constraints command to constrain a primary input pin to a value that removes the X:

add_input_constraints -c0 EN

This allows the tool to consider the CLK signal to be the only clock signal to the clk input of the scan cell.

For further explanation and examples of possible C1 rule violations, refer to the “[Clock Gaters](#)” appendix in the *Tessent Scan and ATPG User’s Manual*.

C2

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Each clock must have a structural path to the clock port of at least one memory element. The application performs this check by a backward tracing from the clock port, SET/RESET port, and read/write enables of all memory elements toward primary inputs. The rule violation occurs if a clock primary input cannot be reached.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. Defining a pin to be a clock, when it does not behave as a clock, is the most usual cause of this error condition. The default handling for this rule violation in all contexts is warning. Failure to satisfy this rule indicates a defined clock cannot capture data, thus reducing test coverage.

When a floating pin is defined as a clock, it is suppressed from C2 violation when the design also defines some internal clocks. This exception prevents false C2 violations for a design with PLL clocking where the external reference clock is disconnected from the netlist in the flat model.

The occurrence message is:

Clock P cannot reach the clock port of any memory element. (C2-1)

P is the pin name of the clock.

The summary message is:

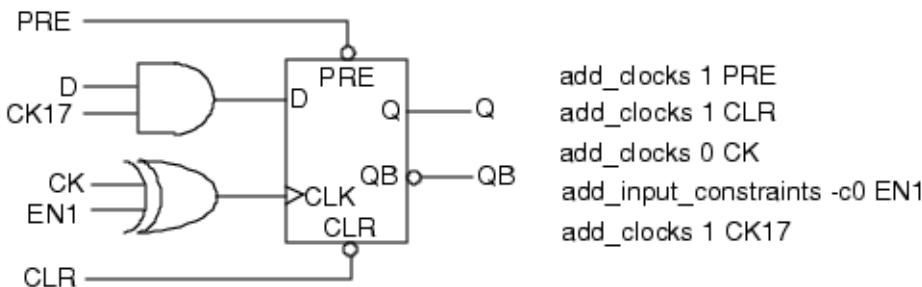
There were N clock rule C2 fails (reachability check).

N is the number of occurrences of rules violation C2.

C2 Rule Violation Example

[Figure 7-11](#) shows an example circuit and circuit setup specified in the tool.

Figure 7-11. C2 Rule Example Circuit



If you run rules checking on this design, you will get a C2 rules violation because while the CK17 signal appears to be a clock (due to its name), it cannot be reached from the clock port or SET/RESET ports of the flip-flop. To fix this problem, add the command:

delete_clocks CK17

Then, you must re-run checks.

C3

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

When a sequential element (or RAM) source and sink are clocked by the same clock and the sink captures data from the source, a potential exists for the captured data to pass through both the source and sink in the same clock cycle. That is, the sink might capture the source's new data (captured in the same cycle) instead of the source's old data (captured in the previous cycle).

[Table 7-2](#) lists four clocking relationships that can produce this problem.

Table 7-2. Clocking that Can Result in a C3 Signal Race

Source	Sink
LS	LS

Table 7-2. Clocking that Can Result in a C3 Signal Race (cont.)

Source	Sink
LS	TE
LE	LS
LE	TE

This condition violates the tool's default assumption that the sink captures only old data from the source, so results in a C3 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Note

 Adequate special handling for most C3 situations is to simply issue “set_split_capture_cycle On” as explained under “[Possible Resolutions](#).”

[Figure 7-12](#) shows an example DFTVisualizer Flat Schematic window display of a circuit segment that produces a C3 violation when the off state of the clock at the PI is defined as a logic 0. (In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups). DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock's trailing edge (TE) due to the inverter. Under certain timing conditions (if captured data propagates through DFF1 to its q output in less than half a clock cycle for example), data will pass through both DFF1 and DFF2 in a single clock cycle.

Note

 The same situation would occur if the clock pin was connected directly (rather than through an inverter) to DFF2, and DFF2 was negative-edge triggered.

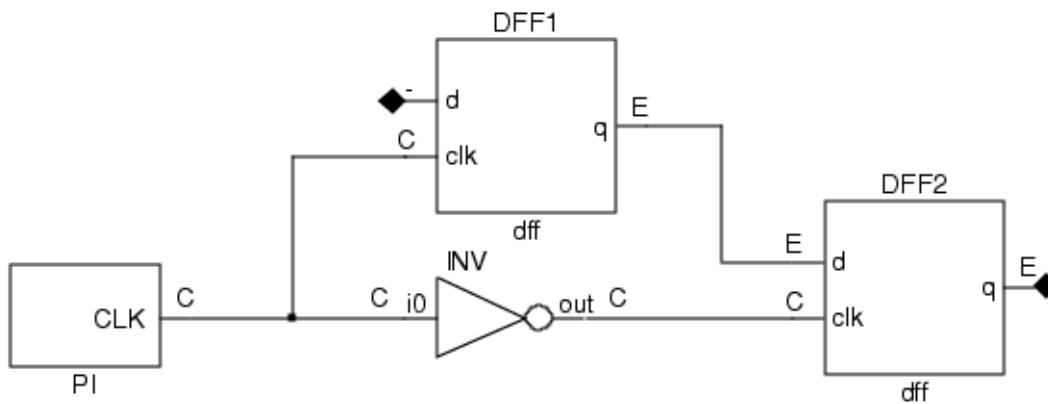
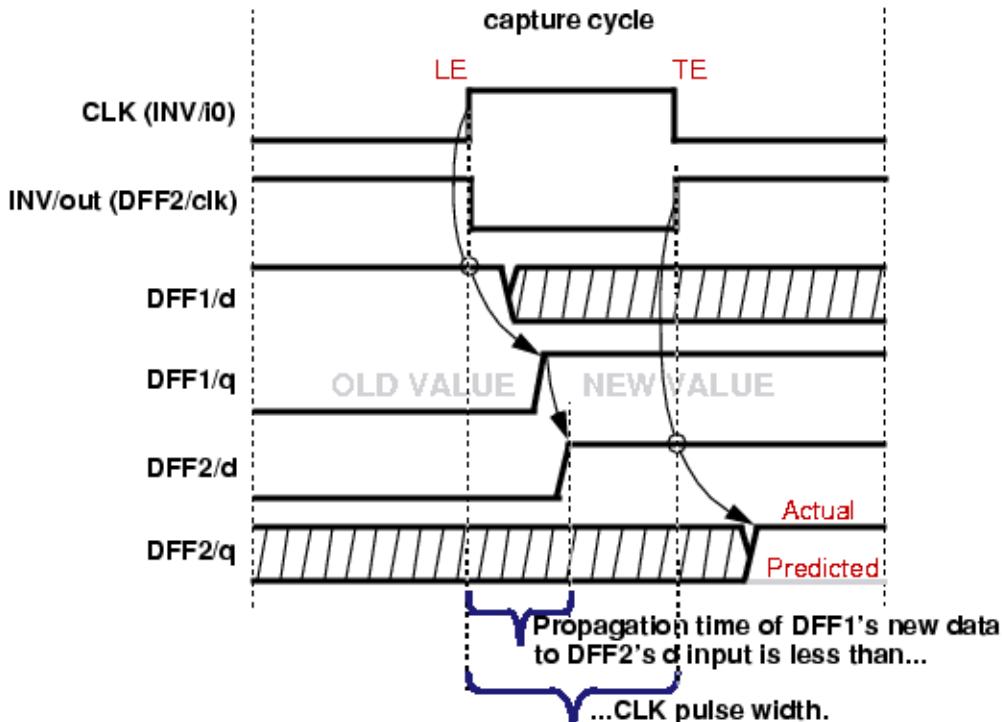
Figure 7-12. C3 Violation Example

Figure 7-13 shows a possible timing scenario for the preceding circuit segment.

Figure 7-13. Example Timing That Allows Sink to Capture Source's New Data



The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones). For an introduction to clock and effect cones, refer to the clock cones and effect cones discussion under “[Clock Signals](#).” The bounds for the clock cones are scan cells and circuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The clock cone stops at read ports of RAMs that have the `read_off` attribute set to hold, and then the effect cone propagates from its outputs.

Note

 In the C3 violation example, the data input for DFF2 is in the effect cone of CLK and its clock input is in the clock cone of CLK as indicated by the E and the C in the Flat Schematic window display of DFTVisualizer.

The rule violation may occur for a clock if one of the following is true:

- The clock input of a DFF state element is in the clock cone, its input data is in the effect cone, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in the clock cone and its input data is in the effect cone.

- The write input of a RAM is in the clock cone and a data input or address input of the associated write port is in the effect cone.
- The read input of a RAM is in the clock cone and an address input of the associated read port is in the effect cone.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the `set_drc_handling` command with the `Atpg_analysis` argument. By default, the rules checker performs a partial ATPG analysis to find potential C3 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

 In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to “[Screening Out False C3 Violations](#).”

Effect on Testability

Failure to satisfy this rule can result in inaccurate test pattern simulation results (simulation mismatches) during verification in a timing-based simulator and failing patterns on the tester.

How to Debug C3 Violations

For C3 violations, the tool transscripts a message similar to the following:

```
// Note: There were N clock rule C3 fails (clock may capture data
// affected by its captured data).
```

N is the number of times a C3 violation occurred.

Use the `report_drc_rules` command to obtain additional information about the violations. For example, to report the occurrence messages for all the C3s, issue “`report_drc_rules C3`.” The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with “C3-” and the occurrence number. For example:

```
report_drc_rules c3-1
// Note: Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
//       Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C3 violation using DFTVisualizer or by issuing commands from the tool’s command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C3 DRC violation using DFTVisualizer, use the following command steps:

1. set_system_mode analysis
2. Choose **Tools > Analyze DRC** from the DFTVisualizer menu and select the desired C3 DRC violation in the dialog box. The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Flat Schematic window.

Tip

 Alternatively, you can issue the `analyze_drcViolation` command with the `rule_id-occurrence#` argument at the tool's command line. For example:
`analyze_drcViolation c3-1`

Note

 In some situations, the tool's analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the `-Interval` switch with the `set_drc_handling` command.

How to Debug from the Tesson Tool Command Line

To view the location of the first occurrence of a C3 DRC violation from the tool command line, use the following example command steps:

1. set_system_mode analysis
2. report_drc_rules C3-*occurrence#*
3. set_gate_report Clock_cone *pin_name*
4. report_gates *source_gate_id* *sink_gate_id*

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

```
report_drc_rules c3-1
// Note: Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
// Source of violation: input 3 of /DFF1 (8)
set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2
// /DFF1 dff
// "IO" I (-) 4-
// "I1" I (-) 3-
// clkI (C) 2-/CLK
// d I (-) 1-/C
// "OUT" O (E) 5-
// /DFF2 (9) dff
// "IO" I (-) 4-
// "I1" I (-) 3-
// clkI (C) 7-/INV/out
// dI (E) 5-/DFF1/q
// "OUT" O (E) 6-
```

Possible Resolutions

Note

 There are special cases where C3 rule violations do not cause problems and can be ignored. Although these cases are rare, be sure you understand how your circuit behaves versus what the tool simulates before deciding to ignore C3s.

There are two methods you can use to assure that valid C3 rule violations do not result in simulation mismatches:

- Split Capture Cycle

The first and preferred method is to enable the `set_split_capture_cycle` command (the tool enables it automatically if you use `create_patterns`). This forces a second evaluation of the cells affected by C3 DRC violations. The additional evaluation allows the new data to propagate and in turn be captured into the sink gate. If additional evaluation is required, a side effect of this command is an increase in simulation run time. If the increase in run time is excessive, you may want to use the other method, capture handling.

Note

 When you use “`set_split_capture_cycle on`,” you will still get C3 rule violations but you can ignore them.

- Capture Handling

The second method of addressing C3 DRC violations is to use the `set_capture_handling` command. A side effect of this command is reduced test coverage if the number of C3s is significantly high. The item to consider is that only one simulation value is used for a

given source. As a result, the sink gates can only capture either new or old data. Given the case where a source drives two sinks and one sink captures old data and the other sink captures new data, one of the sinks must capture X. The advantage of the `set_capture_handling` command is there is no impact on ATPG run time.

Mentor Graphics recommends you use “`set_split_capture_cycle on`.” In cases where the impact on run time is severe and the number of C3 violations is small, then the use of capture handling is recommended. Do not use both methods simultaneously, however.

Screening Out False C3 Violations

For performance reasons, the tool’s default DRC may occasionally report a false C3 violation. There are three additional analyses you can do to screen out these false C3s:

- Use the `set_drc_handling` command with the `Atpg_analysis` option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the `-Mode A` option to the `set_drc_handling` command.

When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

- **Inactive low (IL)**
When the selected clock is low, the clock port of the latch is inactive.
- **Inactive high (IH)**
When the selected clock is high, the clock port of the latch is inactive.
- **Active high slave (AHS)**
When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.
- **Active low slave (ALS)**
When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting with “`set_learn_report On`,” you can then use `report_gates` to report on the individual latches in these categories.

You can screen out false violations of the C3 rule by issuing the `set_drc_handling` command before rules checking. The command usage in this context is:

```
set_drc_handling C3 [-Mode A clock_name]
```

The tool ignores violations of the C3 rule if the following conditions are true:

- The failing latch port is IL or AHS, and the source latch port is IH or ALS.
- The failing latch port is IH or ALS, and the source latch port is IL or AHS.
- All clock, set, and reset inputs of the failing latch are low in the case that all defined clocks are off, the violation source clock is high, and all other clock, set, and reset inputs of the source latch are low.

C4

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When a sequential element (or RAM) source and sink are clocked by the same clock and the output of the source is connected through combinational logic to the clock port of the sink, a potential exists for the source's captured data to alter the clocking of the sink in the same clock cycle. That is, the source's new value (captured in the current cycle) might propagate through the connected logic and affect the clocking of the sink. By default, the tool expects the source's old value (captured in the previous cycle) in the downstream combinational logic.

Table 7-3 lists four clocking relationships that can produce this problem.

Table 7-3. Clocking that Can Result in a C4 Signal Race

Source	Sink
LS	LS
LS	TE
LE	LS
LE	TE

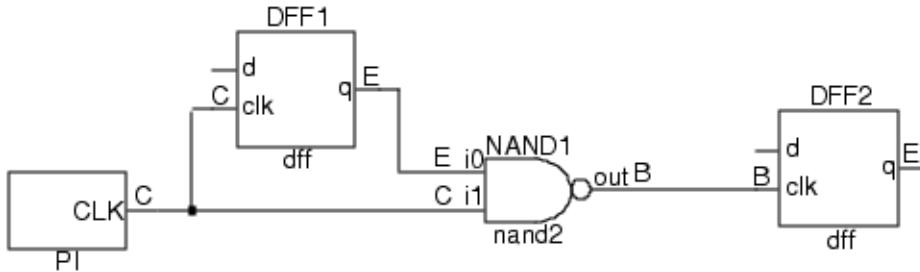
This condition violates the tool's default assumption that the sink's clocking is affected by only old data from the source, so results in a C4 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Note

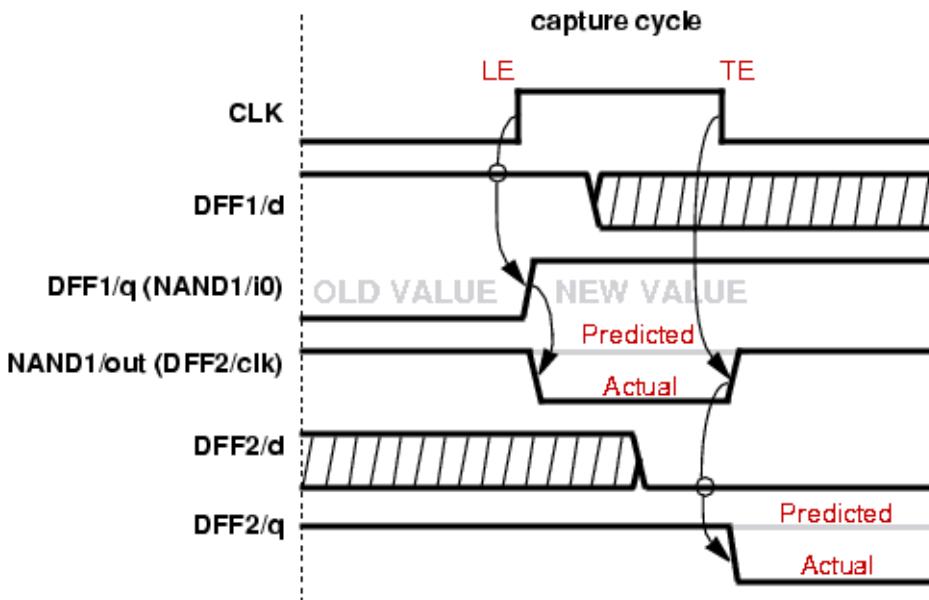
 Adequate special handling for most C4 situations is to simply issue “set_split_capture_cycle on” as explained under “[Possible Resolutions](#).”

[Figure 7-14](#) shows an example DFTVisualizer Flat Schematic window display of a circuit segment that produces a C4 violation when the off state of the clock at the PI is defined as a logic 0. (In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups).

Figure 7-14. C4 Violation Example



[Figure 7-15](#) shows a possible timing scenario for the preceding circuit segment, where DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock’s trailing edge (TE) due to the gating of the CLK signal with the output of DFF1. Notice that the old value of DFF1/q disables the clock input to DFF2. The tool’s default simulation will therefore predict there will be no clock pulse to DFF2 and that it will hold its state as a result. But the leading edge of CLK almost immediately enables the clock gate and DFF2’s clock port does receive a pulse. So rather than holding state as the tool expects, DFF2 captures it’s D input value on the TE of the now enabled clock gate.

Figure 7-15. Example Where Actual Behavior Differs from Tool's Prediction

The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones) and the forward cones of influence for each scan cell influenced by the clock pin (effect cones). For an introduction to clock and effect cones, refer to the clock cones and effect cones discussion under “[Clock Signals](#).” The bounds for the cones are scan cells and circuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states.

Note

In the C4 violation example, the clock input of DFF2 is in both the clock and effect cones of CLK as indicated by the B in the Flat Schematic window display. This is the main difference from a C3 violation in which the clock input of the sink is in the clock cone but *not* in the effect cone and the data input is in the effect cone.

The rule violation occurs for a clock if one of the following is true:

- The clock input of a DFF state element is in both the clock and effect cones, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in both the clock and effect cones.
- The write input of a RAM is in both the clock and effect cones.
- The read input of a RAM is in both the clock and effect cones.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the `set_drc_handling` command with the

Atpg_analysis argument. By default, the rules checker performs a partial ATPG analysis to find potential C4 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

 In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to “[Screening Out False C4 Violations](#).”

How to Debug C4 Violations

For C4 violations, the tool transmits a message similar to the following:

```
// Warning: There were N clock rule C4 fails (clock may be affected by its captured data).
```

N is the number of times a C4 violation occurred.

Use the report_drc_rules command to obtain additional information about the violations. For example, to report the occurrence messages for all the C4s, use:

```
report_drc_rules c4
```

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with “c4-” and the occurrence number. For example:

```
report_drc_rules c4-1
```

```
//Warning: Clock /CLK failed rule C4 on input 3 of /DFF2 (9). (C4-1)
//          Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C4 violation using DFTVisualizer or by issuing commands from the tool’s command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C4 DRC violation using DFTVisualizer, use the following command steps:

1. set_system_mode analysis
2. open_visualizer

Choose the **DRC Violations** tab in the Design Browser window, then open the listed DRC’s by selecting the “+” symbols as needed to find the violation ID of the desired DRC violation. Double clicking the listed violation, or right clicking and selecting **Analyze DRC Violation**

from the popup menu will analyze the violation. The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Flat Schematic window.

Tip

 Alternatively, you can issue the `analyze_drcViolation` command with the `rule_id-occurrence#` argument at the tool's command line. For example:
`analyze_drcViolation c4-1`

Note

 In some situations, the tool's analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the `-Interval` switch with the `set_drc_handling` command.

How to Debug from the Tool Command Line

To view the location of the first occurrence of a C4 DRC violation from the tool command line, use the following example command steps:

1. `set_system_mode analysis`
2. `report_drc_rules C4-occurrence#`
3. `set_gate_report Clock_cone pin_name`
4. `report_gates source_gate_id sink_gate_id`

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

```
report_drc_rules c4-1
// Warning: Clock /CLK failed rule C4 on input 3 of /DFF2 (10). (C4-1)
// Source of violation: input 3 of /DFF1 (9)
set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2
// /DFF1 dff
//     clk    I  (C) /CLK
//     d      I  (-) 1-
//     q      O  (E)  5-/DFF2/d /NAND1/i0
// /DFF2 dff
//     clk    I  (B)  7-/NAND1/out
//     d      I  (-)  5-/DFF1/q
//     q      O  (-)  6-/q2
```

You can see that the DRC violation is occurring because the clock input of /DFF2 is in both the clock and effect cones of /CLK.

Possible Resolutions

Note

-  There are special cases where C4 rule violations do not cause problems and can be ignored. These cases are rare, however, so be sure you understand how your circuit behaves versus what the tool simulates before deciding to ignore C4s.
-

There are two methods you can use to assure C4 rule violations do not result in simulation mismatches:

- Split Capture Cycle

The first and preferred method is to enable the `set_split_capture_cycle` command (the tool enables it automatically if you use `create_patterns`). This forces a second evaluation of the cells affected by C4 DRC violations. The additional evaluation allows the new data to propagate and in turn be captured into the sink gate. If additional evaluation is required, a side effect of this command is an increase in simulation run time. If the increase in run time is excessive, you may want to use the other method, capture handling.

Note

-  When you use “`set_split_capture_cycle on`,” you will still get C4 rule violations but you can ignore them.
-

- Capture Handling

The second method of addressing C4 DRC violations is to use the `set_capture_handling` command. A side effect of this command is reduced test coverage if the number of C4s is significantly high. The item to consider is that only one simulation value is used for a given source. As a result, the sink gates can only capture either new or old data. Given the case where a source drives two sinks and one sink captures old data and the other sink captures new data, one of the sinks must capture X. The advantage of `set_capture_handling` is there is no impact on ATPG run time.

Mentor Graphics recommends you use “`set_split_capture_cycle on`.” In cases where the impact on run time is severe and the number of C4 violations is small, then the use of capture handling is recommended. Do not use both methods simultaneously, however.

Screening Out False C4 Violations

For performance reasons, the tool’s default DRC may occasionally report a false C4 violation. There are three additional analyses you can do to screen out these false C4s:

- Use the `set_drc_handling` command with the `Atpg_analysis` option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the `-Mode A` option to the `set_drc_handling` command.

When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

- **Inactive low (IL)** — When the selected clock is low, the clock port of the latch is inactive.
- **Inactive high (IH)** — When the selected clock is high, the clock port of the latch is inactive.
- **Active high slave (AHS)** — When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.
- **Active low slave (ALS)** — When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting with “set_learn_report on,” you can then use report_gates to report on the individual latches in these categories.

You can screen out false violations of the C4 rule by issuing the set_drc_handling command before rules checking. The command usage in this context is:

set_drc_handling C4 [-Mode A *clock_name*]

The tool ignores violations of the C4 rule if the following conditions are true:

- The source latch port is IL or AHS and all paths from the source latch to the failing latch are blocked when the selected clock is high.
- The failing latch port is IH or ALS and all paths from the source latch to the failing latch are blocked when the selected clock is low.
- The violation source clock input is high and all other clock, set, and reset inputs are low for the source latch.

C5

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A clock pin must not be capable of simultaneously capturing data on multiple ports of the same scannable memory element. The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained value, and initialized non-scan cells are set to their stable state.

The rule violation occurs on a clock pin when multiple clock inputs of a scannable memory element are in the same clock cone and the clock inputs may be on at the same time. The tool performs a mutual exclusivity check to determine if both clock inputs associated with the failure can be active at the same time. If the justification results in a conflict without justifying decision nodes, it will not be considered a rules violation.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a race condition that creates inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Clock P failed rule C5 on input I of N (G) . (C5-1)
```

P is the pin name of the clock, C5 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

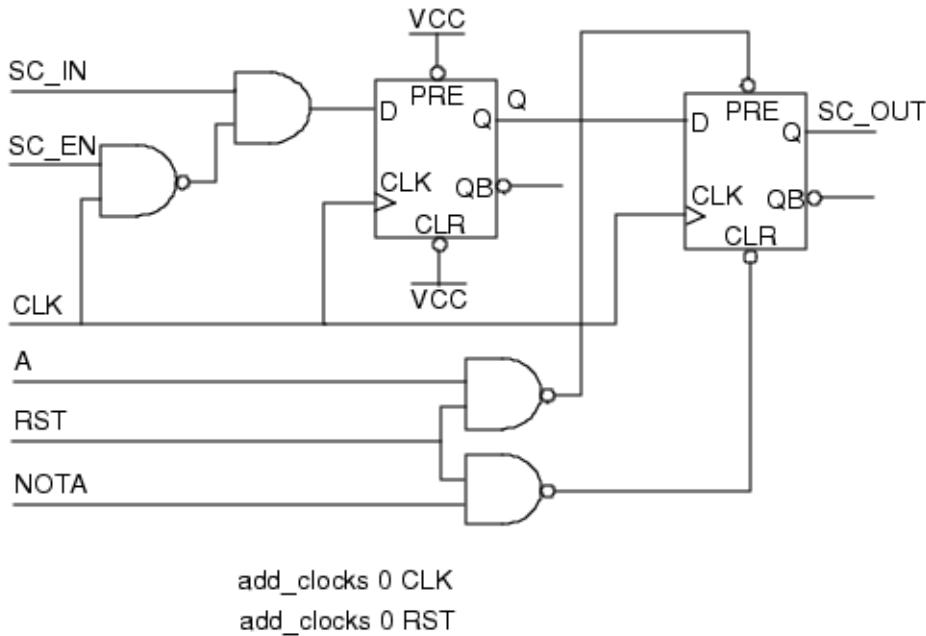
The summary message is:

```
There were N clock rule C5 fails (clock is connected to multiple ports of same latch) .
```

N is the number of occurrences of rules violation C5.

C5 Rule Violation Example

[Figure 7-16](#) shows an example circuit and circuit setup specified in the tool.

Figure 7-16. C5 Rule Example Circuit

If you run rules checking on this design given the setup commands shown, you will get a C5 rules violation. In this design, the RST signal connects to both the PRE and CLR pins of the second flip-flop. If you examine the inputs, you should see that A and NOTA should always have opposite values. If the tool knows this information, it knows that only one of the CLR or PRE signals can be active at any given time. To specify this information and fix the C5 rules violation, add the command:

add_input_constraints a -inv nota

C6

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A clock must not affect data that it is capturing. If it does, a race condition may result that produces inaccurate simulation results.

The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the clock cone are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a clock input of a scannable memory element and its data line are in the same clock cone.

The default handling for this rule violation is warning. When a violation occurs, you can access the clock cone data by the command “set_gate_report drc C6-N,” where N is the c6 fail occurrence number, then issuing the report_gates command for the gate ID number displayed in the occurrence message. This identifies the problem input. By tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

By default, C6 analysis treats clocks in the same synchronous clock group as if they are equivalent. You can use the “set_drc_handling C6 -ignore_synchronous_clock_groups on” command to turn off this behavior.

For more information on synchronous clock groups, see [add_synchronous_clock_group](#).

Note

 Default C6 checking may miss some C6 violations when “set_clock_restriction off” is used. A more conservative analysis that identifies those C6 violations can be enabled by “set_drc_handling c6 -CONSERvative on.” However, we generally recommend not to turn off clock restriction so that the -conservative switch does not need to be used.

The occurrence message is:

```
Clock P failed rule C6 on input I of N (G) . (C6-1)
```

P is the pin name of the clock, C6 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

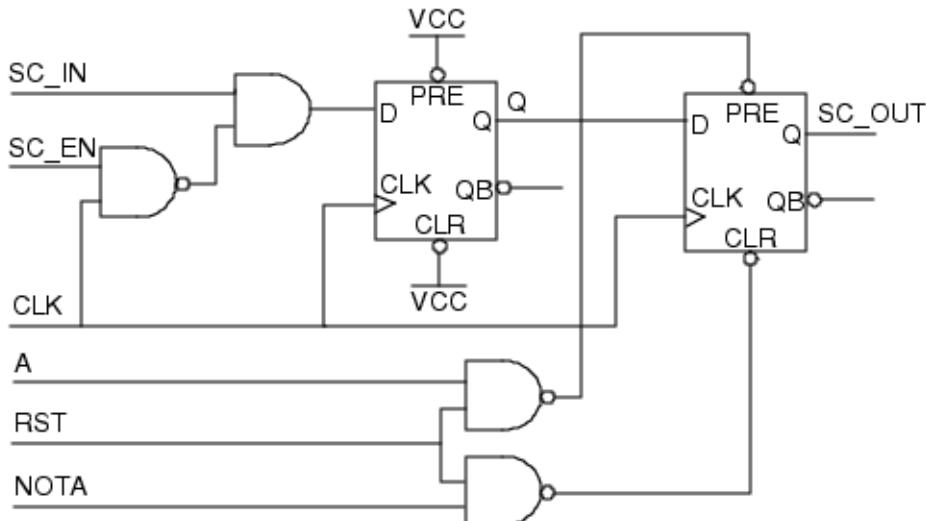
The summary message is:

```
There were N clock rule C6 fails (clock may capture data affected by itself) .
```

N is the number of times a C6 rule violation occurred.

C6 Rule Violation Example

[Figure 7-17](#) shows an example circuit and circuit setup specified in the tool.

Figure 7-17. C6 Rule Example Circuit

```
add_clocks 0 CLK
add_clocks 0 RST
```

If you ran rules checking on this design, given the setup commands shown, you would get a C6 rules violation. The default handling for this rule violation is warning. In this design, the CLK signal goes to both the CLK and D inputs of the first flip-flop. Thus, data can be captured in this flip-flop that may be affected by the capturing clock. To prevent the CLK signal from influencing the data, add the command:

add_input_constraints -c0 SC_EN

Constraining the SC_EN signal to 0 ensures that changes in the clock will not change the data.

For ATPG, you can handle C6 rule violations by issuing a “set_clock_off_simulation on” command prior to creating patterns. See the set_clock_off_simulation command description for additional information.

In some designs, different flip-flops may capture the off state and on state of the clock causing the simulation to fail. In this case, you can use one of the following two commands to mask the signal creating the C6 violation. Note the add_cell_constraints command is more restrictive than the set_simulation option command.

add_cell_constraints -Drc C6

set_simulation_options -C6_mask_races ON

For more information, see the add_cell_constraints and set_simulation_options commands.

C7

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Each clock input (not including set and reset lines) of a scan or non-scan cell memory element must be capable of capturing data when a single clock primary input line is on and all other clocks are off. It is acceptable that this may require placing values on non-clock primary inputs or scan cells. The application performs this check using the simulated values that result when one clock is set to X, all other defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs when a clock input of a scan cell always remains off.

The default handling for this rule violation is warning. Failure to satisfy this rule indicates a scan cell clock input cannot capture data, resulting in some loss of test coverage.

The occurrence message is:

```
Clock input I of N (G) cannot capture data with a single clock on. (C7-1)
```

I is the input number, N is the instance name, G is its gate index number, and C7 is the rule ID number.

The summary message is:

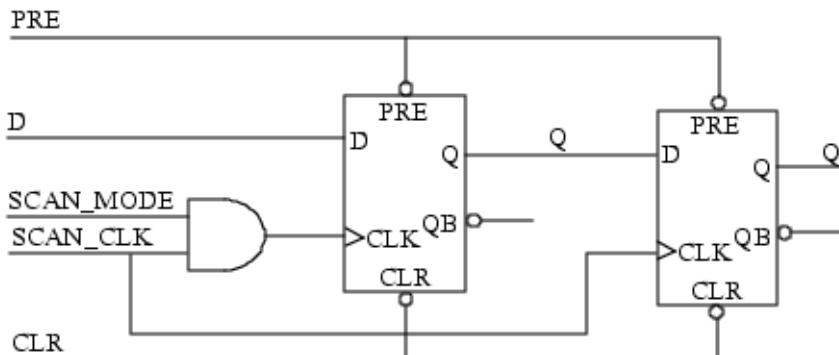
```
There were N clock rule C7 fails (scan cell capture ability check).
```

N is the number of occurrences of rules violation C7.

C7 Rule Violation Example

[Figure 7-18](#) shows an example circuit and circuit setup specified in the tool.

Figure 7-18. C7 Rule Example Circuit



```
add_clocks 1 PRE CLR  
add_clocks 0 SCAN_CLK  
add input constraints -c0 SCAN MODE
```

If you run rules checking on this design given the setup commands shown, you will get a C7 rules violation. This type of error commonly occurs when incorrect clock or set/reset gating occurs in the design. This design constrains the SCAN_MODE signal to a constant 0 during ATPG. This constraint prevents the first flip-flop from ever being clocked, and from ever capturing data. To fix this problem, delete the pin constraint:

delete_input_constraints -all

C8

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report drc rules: Supported

You may not directly connect a clock to a primary output (PO). The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values, and initialized non-scan cells are set to their stable states. The rule violation occurs when a primary output is in the clock cone.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in the occasional usage of a different type of scan pattern, in which the tool observes only the POs directly connected to clocks. There will be no loss of test coverage or risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to `error_pattern` and using the `report_gates` command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

Primary output P is connected to clock C. (C8-1)

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

There were N clock rule C8 fails (PO connected to a clock line).

N is the number of occurrences of rules violation C8.

C8 Rule Violation Example

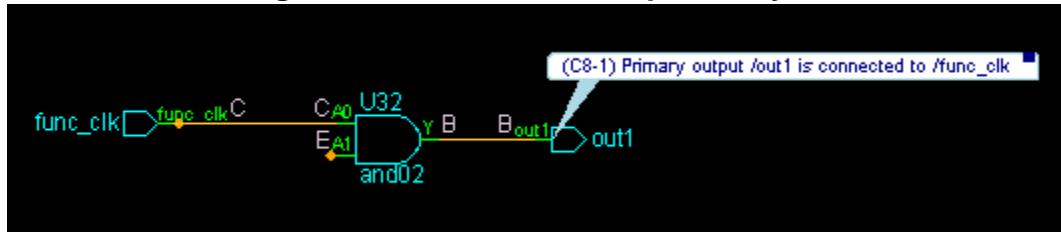
The tool setup for the clocks is:

```
add_clocks 0 func_clk
```

If you run rules checking on this design given the setup command shown, you will get a C8 rule violation. This type of violation is not usually a problem.

Figure 7-19 shows the results of the C8 DRC analysis in DFTVisualizer.

Figure 7-19. C8 Rule Example Analysis



In this example, C8 is caused by the connection of the primary output, out1, through U32 to a clk, func_clk. An acceptable solution is to accept this warning, or turn it off with the command:

```
set_drc_handling c8 ignore
```

C9

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Data captured by any clock with a direct path (through combinational logic only) to a primary output must not affect the direct path to a primary output of that same clock. The application performs this check by determining the forward cone of influence for a clock pin (clock cone) and for each scannable memory element influenced by the clock pin (effect cone). The bounds

for the cones of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a primary output is in both the clock cone and the effect cone.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a small loss in test coverage. There will be no risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
PO P path from clock C is gated by scan cell that uses same clock. (C9-1)
```

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

```
There were N clock rule C9 fails (PO connected to a clock line gated by
scan cell that uses same clock).
```

N is the number of occurrences of rules violation C9.

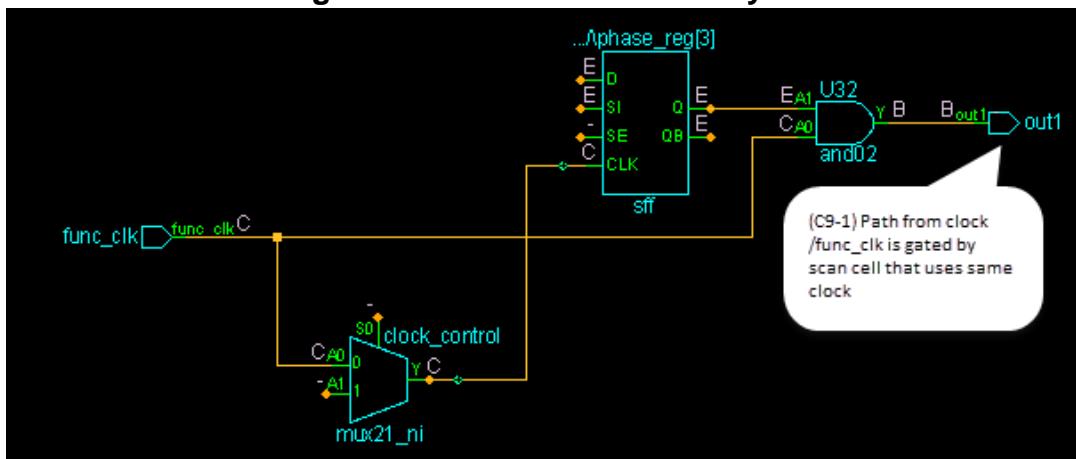
C9 Rule Violation Example

The tool setup for the clocks is:

```
add_clocks 0 func_clk
```

If you run rules checking on this design given the setup command shown, you will get a C9 rule violation. The results of performing DRC Analysis in DFTVisualizer on the C9 DRC in this example is shown in [Figure 7-20](#)

Figure 7-20. C9 Rule DRC Analysis



This type of violation is not usually a problem. However, a C9 violation can result in reduced coverage because it may introduce sequential effects into the generated clock patterns. In this case, the violation is at the out1 line. The func_clk signal can affect the data of the flip-flop and the gate at the output of the scan cell. An acceptable solution is to accept this warning, or turn it off with the command:

```
set_drc_handling c9 ignore
```

C10

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

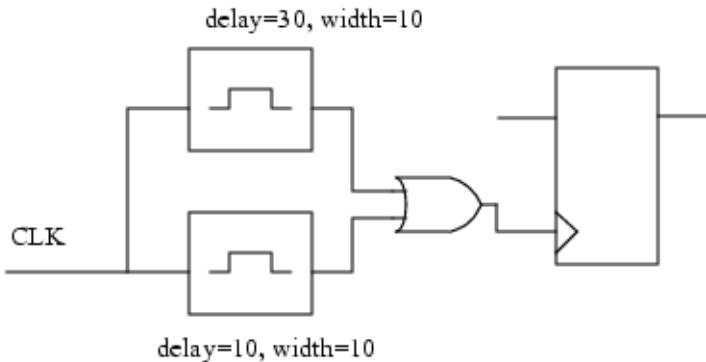
A sequential element (latch or flip-flop) can only be clocked once in any one pattern cycle. The tool will only pulse a clock primary input once in each cycle, or apply a clock procedure only once, therefore a failure of this rule implies that the circuit generates two or more internal pulses from a single clock pulse or clock procedure.

The default handling of this rule violation is error. Failure to satisfy this rule will result in creation of patterns which are likely to be incorrect and to fail both in verification and on the tester.

The occurrence message is:

```
Cell c might capture more than once by applying clock ck. (C10-1)
```

Figure 7-21 shows an example circuit which will generate a C10 error.

Figure 7-21. C10 Rule Example Circuit

In order to remove the violation, the circuit must be modified. Test logic can be added to block the path from one pulse generator to the OR gate during test mode.

Note

It is also possible to violate this rule by OR-ing together two clocks which are pulsed at different times in a clock procedure.

C16

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When switching from the load_unload phase to the capture phase, sometimes the loading values of scan cells can be disturbed due to pin constraints specified in the capture phase. The C16 clock rule checks for this situation and issues a violation if it occurs.

For example, consider a DFF scan cell with two clock ports: the first port is used for scan loading and the second is used for normal operation. Assume the second port's clock signal (whose off state is defined as 1) is ANDed with an enable signal to produce the actual clock input value. If in the load_unload procedure the enable is forced to 0, the load value cannot be disturbed by the second clock port during scan loading. However, if you issued an add_input_constraints command in setup mode to set the enable signal to 1, the scan load value can be disturbed at the start of the capture cycle when switching from load_unload to capture. This is because the clock input at the second port will change from 0 to 1 when the enable changes from 0 to 1 as a result of the pin constraint.

Note

The C16 rule check will ignore an X at a scan cell's clock port, as it is possible to assign a proper value at the PIs to keep the scan cell stable at the beginning of the capture phase. Xs on scan cell clock ports will, however, cause C1 violations.

The default handling for rule C16 is error. Failure to satisfy this rule may result in simulation mismatches during verification.

To debug a C16 rule violation, issue a “set_gate_report_error_pattern” command, then use report_gates for the gate instance named in the C16 occurrence message.

Caution

 If you choose to ignore C16 violations (not recommended), you should add a CX cell constraint to each scan cell where a violation occurred—in order to avoid simulation mismatches. Be aware that each cell you constrain in this manner reduces test coverage. You can obtain the instance name of the scan cell from the occurrence message and use the add_cell_constraints command to apply the constraint to the cell’s output pin. For example:
report drc rule c16
// Warning: The load value of scan cell /dff2/dff1/Q_reg (47) is disturbed by clock input 3 at the beginning of the capture phase when pin constraints are forced. (C16-1)
add_cell_constraints /dff2/dff1/Q_reg/Q CX

The occurrence message is:

The load value of scan cell N is disturbed by clock input I at the beginning of the capture phase when pin constraints are forced. (C16-1)

N is the instance name of the gate, I is the gate input number of the clock line, and C16 is the rule ID number.

The summary message is:

There were N clock rule C16 fails (scan cell load value disturbed at beginning of capture phase when pin constraints are forced).

N is the number of occurrences of rules violation C16.

C17

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore for patterns -scan context. Warning for dft -scan and dft -test_points contexts.

report_drc_rules: Supported

The default handling for the C17 rule in patterns -scan context is Ignore. Setting the handling to Ignore bypasses the C17 rule completely, cutting run time.

The default handling for the C17 rule in the dft -scan and dft -test_points contexts is Warning. When handling is set to Warning, all internally-driven glitches are identified (as C17 violations) and are also remodeled as possible X sources during DRC and pattern generation.

You can change the DRC handing for C17 in any of the supported contexts. In addition to setting handling to Ignore and Warning, you can also set handling to Error or Note. Setting handling to Error reports the first violation and then aborts the system mode transition. When handling is set to Note, all internally-driven glitch sources in a design are identified (as C17 violations), but they are not remodeled and will maintain a constant value during DRC and pattern generation.

An internally driven glitch source consists of either a two-input AND gate, or a two-input OR gate where both inputs are driven from a common source (either directly or through buffers and inverters), such that one path is inverting and the other path is not. This type of circuit maintains the same constant output value regardless of the state of the driving node, but the output will probably contain a glitch when the driving node changes state.

With this rule, gates are identified as internally-driven glitch sources if they are both driven by a sequential element and also drive the clock or set/reset line of a sequential element.

Note

 Internally-driven is defined as a glitch generator driven from a sequential element, and not from a primary output.

Remodeling these internally-driven glitch sources as a constant value (TIE0 or TIE1) may cause failures during scan chain tracing. For this reason, if handling is set to Warning, C17 identifies these internally-driven glitch sources and remodels them as an X source (TIEX gate).

You can use the `report_drc_rules` command to display a list of the internally-driven glitch sources identified in your design.

C19

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The C19 rule checks any clock gating structure where a leading edge (LE) or trailing edge (TE) flop is gating a clock path that drives any state elements that can capture a value at the same edge.

Note that the violation can lead to simulation mismatch due to a race condition between the clock enable path and the data ports of the controlled state elements. You should investigate and fix C19 violations by changing the design or adding constraints to prevent the clock enable signal from changing the value while the data ports of the controlled state elements are capturing data.

To debug a C19 rule violation, issue the `analyze_drcViolation` command to display the following in the Flat Schematic window:

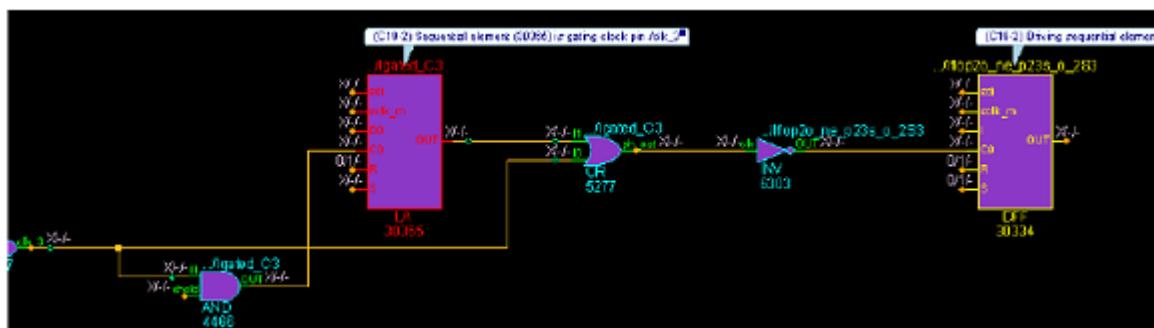
- path from the clock enable flop to the driving flop with the violation
- path from clock pin to enable flop
- constraint value
- clock enable flop and driving flop with the violation

Example of C19 Violation

The following example shows a design with 10 C19 violations:

```
SETUP> set_system_mode analysis
...
// 10 sequential elements are gating clock path and driving sequential elements on the same
// clock edge which can create a race condition. (C19)
...
ANALYSIS> report_drc_rules c19-2
// Warning: Active-high latch /lbasic_gates/l gated_C3/ (30365) is gating clock pin /clk_3 (17)
// and driving some leading-edge flops which can create a race condition. (C19-2)
// The first traced sequential element driven by /lbasic_gates/l gated_C3/ (30365) that can
// create a race condition is /lbasic_gates/l flop2o_ne_p23s_o_2S3/ (30334).
ANALYSIS> report_drc_rules
...
C19: #fails=10 handling=warning/atpg (clock gater race condition check)
...
ANALYSIS> analyze_drcViolation C19-1
```

This command produces the following display in the Flat Schematic window:



C21

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

The C21 DRC detects when multiple active shift clocks converge at the same combinational gate.

When two or more active shift clocks converge at a combinational gate pin (except for MUX gates) and there is an unblocked path to at least one sequential element, the C21 DRC violation is issued. C21 identifies the gate pin location and lists the clocks that converge at that gate pin.

The default handling of C21 is Note, but it is allowed to be set to Warning, Ignore, or Error.

The occurrence message is:

```
// Error: At least 2 active shift clocks (/clk1, /clk2) converge at gate
'gate13/Y' (33) during shift. (C21-1)
```

The summary message is:

```
C21: #fails=1 handling=error/verbose (multiple shift clocks converge at
same gate)
```

C22

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The C22 DRC reports a violation when the tool identifies asynchronous cross domain clock paths where the launch and capture clocks are forced to pulse, which could lead to simulation mismatches.

C22 checks a pair of clocks, one of which must be pulse-always or pulse-in-capture, whether there is any interaction between the pair and they are not defined within a synchronous clock group.

C22 also checks if any interaction exists between an internal clock of a clock_control procedure and its source clock when the clocks are not defined within a synchronous clock group. In such a case, C22 reports a violation.

When you issue a [read_profile](#) command that adds, deletes, or modifies the clock_control procedures, the C22 DRC will be re-analyzed and any false paths derived by C22 will be updated.

[Figure 7-22](#) and [Figure 7-23](#) include examples of cross clock domain paths.

The default C22 DRC handling is Warning, and can be set to Error, Warning, Note, or Ignore.

The occurrence message is one of these:

```
// Warning: Pulse-always clock '/clk1' (7) interacts with clock domain
// '/clk2' (8) and may cause simulation mismatches. Use "report_clock_domains
// -race_points 7 8" to see the clock interactions. (C22-1)

// Warning: Pulse-in-capture clock '/CLK2' (2) interacts with clock
domain '/CLK1' (1) and may cause simulation mismatches. Use
"report_clock_domains -race_points 2 1" to see the clock interactions.
(C22-1)

// Warning: Clock control source clock '/CLK1' (1) interacts with clock
domain '/CLK1_GATED' (25) and may cause simulation mismatches. Use
"report_clock_domains -race_points 1 25" to see the clock interactions.
(C22-1)
```

The summary message is:

```
// Warning: There were 2 C22 violations (interacting clocks not in
synchronous clock group but forced to pulse concurrently)
```

Unless the C22 handling is set to Error, the tool adds a false path between the clock pair to prevent simulation mismatch due to C22. If set to Ignore, no false paths will be added, because no C22 is flagged. The `-mask_interactions` and `-interactions` switches of the `set_drc_handling` command allow you to disable the false path addition and to determine the type of clock interaction for C22.

You can report the number of sequential elements impacted by each C22 violated clock pair using this command in ANALYSIS mode:

```
report_clock_domains -race_points clk1 clk2

//      /clk_gater_2/pipe_dff1_1/ (98) scan
// 1 state element causes clocks '/clk1' (7) and '/clk2' (8) to be
incompatible.
```

With the information returned by the `report_clock_domains` command, you can trace backwards to locate the clocks using the `report_gates` command or within DFTVisualizer. [Figure 7-22](#) shows the most basic C22 DRC violation example. The C22 violation is between CLK1 (a pulse-always clock) and CCB2 because of the cross domain path from SFF1 to SFF2.

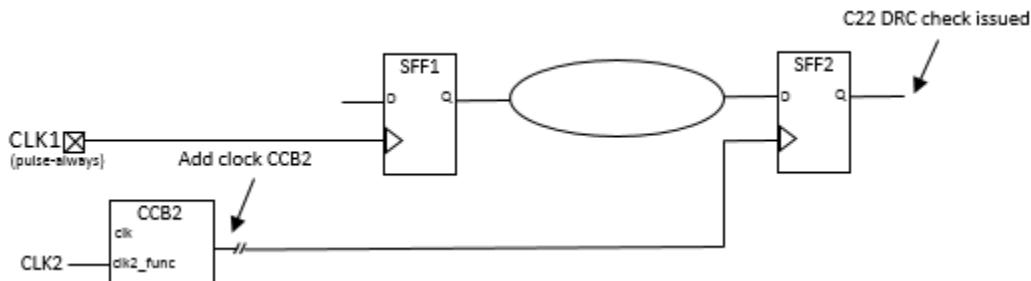
Figure 7-22. C22 DRC Example Design 1

Figure 7-22 also demonstrates the interaction of an internal clock of a `clock_control` procedure (CCB2) and its source (CLK1) when the clocks are not defined within a synchronous clock group. This is the `clock_control` procedure for this example:

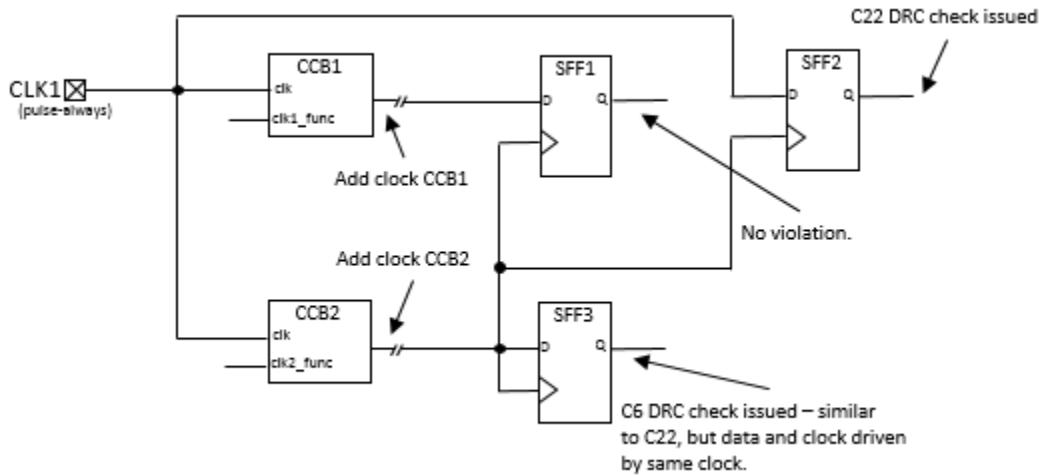
```
clock_control CCB2 =
  source_clock CLK1;
  atpg_cycle 0 =
    condition /sff2_CC/Q 1;
  end ;
  atpg_cycle 1 =
    condition /sff1_CC/Q 1;
  end;
end;
```

When a source and internal clock pair are defined in a `clock_control` procedure, it is not necessary for the source (CLK1, in this case) to be defined as a pulse-always or pulse-in-capture for C22 to trigger a violation.

Figure 7-23 shows another C22 DRC violation example. The C22 violation is between CLK1 and CCB2 and their interaction on SFF2. SFF1 does not violate C22 because it is driven by the added clocks CCB1 and CCB2, neither of which is pulse-always nor pulse-in-capture.

A C6 DRC is issued for SFF3 and is shown in this example to demonstrate that while C22 is similar to C6, C22 flags different clocks and C6 involves the same clock.

Figure 7-23. C22 DRC Example Design 2



Possible Resolutions For C22 Violation

One possible resolution is to change the design to eliminate the cross domain path. If the clocks are synchronous, another solution would be to use the `add_synchronous_clock_group` command to place the clock pair in the same synchronous clock group in Setup mode.

C23

Category: Clock

Contexts Supported: `dft -edt`, `dft -logic_bist`, `patterns -scan`, `patterns -scan_diagnosis`

Default Handling: Warning

`report_drc_rules`: Supported

This rule checks if a scan cell cannot capture fault effects because its clock port is constrained to 0 or 1.

The information provided by C23 helps you debug low fault coverage caused by clocking issues. To check for C23, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port, except those meant to be used only for scan shift, and checks for these conditions:

- Are the clock ports of level-sensitive latches constrained to 0?
- Are the clock ports of edge-triggered DFFs constrained to 0 or 1?

If all the clock ports of a scan cell satisfies either of these conditions, a C23 violation is reported because none of its clock ports can be turned on.

To minimize the number of violations reported, the tool traces from rule violation clock ports to a common source point, which can be a primary input or internal pin. The violating sources are

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C23

```
// Warning: Primary input '/clk (4)' has a constrained value of 0. This
// prevents capture by 32 scan cell elements, one of which is '/inst/sdff1
// (880)'. (C23-1)

// Warning: Internal pin '/inst1/udp/A (567)' has a constrained value of
// 1. This prevents capture by 18 scan cell, one of which is '/inst/sdff2
// (990)'. (C23-2)
```

The default handling of C23 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
// Warning: There are N clock rule C23 fails (scan cell elements cannot
// capture or launch transition because clock ports are constrained).
```

To debug C23, use the “[set_gate_report -constrain_value on](#)” command to view the constrained values for the clock ports of affected scan cells.

C24

Category: Clock

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a scan cell cannot capture fault effects because its clock port cannot be set to an active state. A scan cell must first pass the C23 DRC in order to be checked by C24.

The information provided by C24 helps you debug low fault coverage caused by clocking issues. To check for C24, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port, except those meant to be used only for scan shift, and checks these for conditions:

- The constrained value of the clock ports of level-sensitive latches is X, but its forbidden value is 1, or the clock cannot be justified to 1.
- The constrained value of the clock ports of edge_triggered latches is X, but its forbidden value is either 0, 1, or both, or the clock cannot be justified to both 0 and 1.

If all the clock ports of a scan cell satisfies either of these conditions, a C24 violation is reported because none of its clock ports can be turned on.

To minimize the number of violations reported, the tool traces from rule violation clock ports to a common source point, which can be a primary input or internal pin. The violating sources are

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C24

```
// Warning: Primary input '/clk (4)' has a value of X and cannot be
justify to 1. This prevents capture by 32 scan cell elements, one of
which is '/inst/sdff1 (880)'. (C24-1)

// Warning: Internal pin '/inst1/udp/A (567)' has a value of X and cannot
be justified to both 0 and 1. This prevents capture by 18 scan cell
elements, one of which is '/inst/sdff2 (990)'. (C24-2)
```

The default handling of C24 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
// Warning: There are N clock rule C24 fails (scan cell elements cannot
capture or launch transition because clock ports cannot be fully
controlled).
```

To debug C24 violations, use the “[set_gate_report -constrain_value on](#)” command to look at the constrained values on clock ports of affected scan cells.

C25

Category: Clock

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a scan cell cannot capture fault effects because its asynchronous controls (SET and RESET ports) cannot be set to the off state.

To check for C25, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes the set and reset ports of every scan cell and checks for these conditions:

- If the port’s constrained value is 0, it passes.
- If the port’s constrained value is a 1, it fails because the scan cell is always setting or resetting and is unable to capture fault effects through its data port.
- If the ports constrained value is X, and its forbidden value is 0 or cannot be justified to 0, it fails, because the port cannot be at its off state. Otherwise, it passes, because ATPG can justify the port to its off state and allow the cell to capture.

To minimize the number of violations reported, the tool traces back constant or X paths to a common source point, which can be a primary input or internal pin. The violating sources are

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C25

```
// Warning: Primary input '/clk (4)' is has a constrained value of 1. As a result, set or reset ports cannot be turned off for 32 scan cell elements, one of which is '/inst/sdff1 (880)'. (C25-1)
```

```
// Warning: Internal pin '/inst1/udp/A (567)' has a value of X and cannot be justified to 0. As a result, set or reset ports cannot be turned off for 18 scan cell elements, one of which is '/inst/sdff2 (990)'. (C25-2)
```

The default handling of C25 is Warning. It is performed during system mode transition from Setup to Analysis. This is the summary message:

```
// Warning: There are N clock rule C25 fails (scan cell elements' set/reset port cannot be justified to off state).
```

To debug C25 violations, use the “[set_gate_report -constrain_value on](#)” command to look at the constrained values on set and reset ports of affected scan cells.

Scan Cell Data Rules (D Rules)

The DFT tools check scan cells to ensure they are able to properly control and observe their data. You may select the handling of these scan cell data rules to be error, warning, note, or ignore.

The following subsections describe these rules.

Tip

 In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see “[How to Report Gate Data](#).”

D1.....	2732
D2.....	2737
D3.....	2738
D4.....	2738
D5.....	2739
D6.....	2740
D7.....	2741
D8.....	2742
D9.....	2743
D10.....	2744
D12.....	2746

D1

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Checks for the possible disturbance of data values loaded or captured into scan cells.

During the application of the test procedures, if other circuitry disturbs the data values loaded or captured into scan cells, those cells cannot be controlled or observed. The tool performs this check using the simulated values of each time period of the test procedures. A violation occurs if any clock input (including set and reset lines) of any scan cell allows data capture at an inappropriate time, in which case the control value loaded or the capture value unloaded from the scan cell cannot be trusted. Common sources of D1 violations are undefined clocks, problems in the test procedure file, or possibly design errors.

Effect on Testability

Failure to satisfy this rule can result in simulation mismatches when you verify the test patterns in a timing-based simulator. If the violations occur during the shift procedure, you may see related simulation mismatches in the serial pattern test bench only, not the parallel test bench.

If the number of D1 violations is small compared to the number of scan cells in the design, you may choose to handle this rule violation as a warning by issuing the `set_drc_handling D1` Warning command prior to DRC. When “`set_drc_handling d1`” warning is in effect for DRC, the tool automatically applies an XX cell constraint to each D1 failing scan cell. This prevents the mismatches, but may reduce test coverage slightly. See the `add_cell_constraints` command description for information about the XX cell constraint.

Note

 If “`set_drc_handling d1`” note is in effect for DRC, the tool will *not* apply any XX cell constraints for D1 violations, which ensures test coverage is not affected. However, you should use “note” handling only for cell architectures you know will not produce mismatches.

If the number of D1 violations is large, you should isolate and resolve the source of the failure(s) to avoid a major impact on test coverage.

How to Debug D1 Violations

The occurrence message is:

```
// N (G) disturbed during time T of P procedure. (D1-1)
```

N is the instance name of the scan cell memory element, G is the gate ID number, T is the time period, and P is the group test procedure name.

The summary message is:

```
// There were N occurrences of scan cell disturbs. (D1)
```

N is the number of occurrences of rules violation D1.

Use the `report_drc_rules` command to obtain additional information about the violations. For example, to view the occurrence messages for all D1s, use:

```
report_drc_rules d1
```

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with “D1-” and the occurrence number. For example:

```
report_drc_rules d1-1
```

```
// Error: /dataout/reg_q_0_ (373) disturbed during time 0 of grp1
load_unload procedure. (D1-1)
```

When a D1 error occurs, you can access the simulated values at the gate where the error occurred by issuing the command, `set_gate_report Error_pattern`, then using the `report_gates` command to report the gate whose ID number is displayed in the error message. In the resultant display, you can identify the clock input not held at its off state. By tracing back from this input, you can usually identify how to correct the problem.

You can debug a specific occurrence of a D1 violation using DFTVisualizer or by issuing commands from the tool's command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a D1 violation using DFTVisualizer, use the following command steps:

1. `open_visualizer`
2. `set_gate_level Primitive`

Choose the **DRC Violations** tab in the Design Browser window, then open the listed DRC's by selecting the “+” symbols as needed to find the violation ID of the desired DRC violation.

Double clicking the listed violation, or right clicking and selecting **Analyze DRC Violation** from the popup menu will analyze the violation. DFTVisualizer will display the scan cell affected by that occurrence.

Tip

 Alternatively, you can issue the `analyze_drcViolation` command with the `rule_id-occurrence#` argument at the tool's command line. For example, to display the first occurrence of a D1 violation: `analyze_drcViolation d1-1`

Once the offending scan cell is displayed, trace back from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the `add_clocks` command, is correct.

How to Debug from the Tool Command Line

To view the location of a D1 violation from the command line, use the following example command steps:

1. `set_gate_report Error_pattern`
2. `set_gate_level Primitive`
3. `report_drc_rules D1-occurrence#`
4. `report_gates offending_gate's_id#`

The following transcript excerpt shows an example of the use of this command sequence:

```
set_gate_report error_pattern
set_gate_level primitive
report_drc_rules d1-1
// ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//          load_unload procedure. (D1-1)
report_gates 373
// /dataot/reg_q_0_ (373) DFF
//   "S"    I  (0)  43-
//   R     I  (0)  157-
//   CLK   I  (1)  313-/dataot/ix104/Y
//   "D0"   I  (X) 255-
//   "OUT"  O  (X) 127- 128-
```

You would then trace back from the CLK input to try to determine why it is in an active state, which disturbed the cell's value.

In some of the more complex cases, the problem can be a clock disturb early in the capture cycle. In these cases, the error report does not provide enough detail to isolate the failure. To get more information, you can use the following command steps:

1. set_gate_report Drc_pattern State_stability
2. set_system_mode analysis
3. report_gates *offending_gate's_id#*

The following transcript excerpt shows an example of the use of this command sequence:

```
set_gate_report drc_pattern state_stability
// Creating schematic for 3 instances (0 were compacted).
set_system_mode analysis
// -----
// Begin scan chain identification process, memory elements = 56.
// -----
// Reading group test procedure file fast.testproc.
// WARNING: Pin tclk is used for both write control and pulse clock
// Simulating load/unload procedure in grp1 test procedure file.
// Chain = chain1 successfully traced with scan_cells = 12.
// Chain = chain4 successfully traced with scan_cells = 12.
// Chain = chain3 successfully traced with scan_cells = 12.
// Chain = chain2 successfully traced with scan_cells = 12.
// 48 scan cells have been identified in 4 scan chains.
// Longest scan chain has 12 scan cells.
// WARNING: 8 edge-triggered clock ports set to stable high. (D7)
// ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//        load_unload procedure. (D1-1)
// ERROR: Rules checking unsuccessful, cannot exit SETUP mode.
report_gates 373
// /dataot/reg_q_0_ (373) DFF
//           (ts) (ld) (shift) (cap) (stbl)
// "S"   I  ( 0 ) ( 0 ) (000~0) (000) (    0)  43-
// R    I  ( 0 ) ( 0 ) (000~0) (XXX) (    X)  157-
// CLK  I  ( 1 ) ( 1 ) (101~1) (XXX) (    X)  313-  /dataot/ix104/Y
// "D0"  I  ( X ) ( X ) (XXX~X) (XXX) (    X)  255-
// "OUT" O  ( X ) ( X ) (XXX~X) (XXX) (    X)  127-  128-
```

The item to notice is the clock values during the capture cycle. In this case the value of the clock is (XXX). This indicates the clock is not at its off state at the beginning or end of the capture cycle. The correct values should be either:

- (0X0)—For a leading edge device with an off state of 0 for the clock, or
- (1X1)—For a trailing edge device with an off state of 1 for the clock

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. To change the defined off state, first delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

If the clocks are working correctly, examine the test procedure file. By tracing through the design, you may find an incorrect constraint or force value that can be fixed in the test procedure file.

Note

 If you make any changes in the test procedure file, execute the read_procfile command with the correct filename before re-checking the design rule checks.

D2

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

During system data capture, if the scan path from a MASTER or SLAVE element to its COPY is sensitized, it must be unique. The tool performs this check by comparing the inputs of the COPY with its associated memory element. A violation occurs if there are multiple paths from the associated memory element (MASTER or SLAVE) to the COPY and these paths can be sensitized at the same time.

The application checks for the following conditions:

- The value on the data line of the COPY element must be able to be propagated back to its associated memory element along the scan path when constrained pins are set.
- The scan path, when sensitized from the associated memory element to the COPY, must be unique.
- The COPY and its associated memory element may have only a single clock port.
- For a COPY and its associated memory element, all non-tied clock, set, and reset inputs must have the same or equivalent source.

Note

 DRC does not consider it a D2 violation if a COPY element can capture a value through a different path other than from its associated memory element during system capture.

Failure to satisfy this rule usually reduces test coverage. You can avoid the coverage loss by using the command, set_split_capture_cycle On.

The occurrence message is:

```
COPY N (G) failed data capture check. (D2-1)
```

N is the instance name of the COPY memory element, and G is the gate ID number.

The summary message is:

```
N COPY scan elements failed data capture check. (D2)
```

N is the number of occurrences of rules violation D2.

D3

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

For all scan cells that contain a SLAVE, the **master_observe** procedure must propagate the data value of the MASTER memory element to the SLAVE. The application performs this check using the simulated values of each time period of the **master_observe** procedure to trace back from the SLAVE to the MASTER. The rule violation occurs if the **master_observe** procedure does not properly sensitize the path between the SLAVE and MASTER.

The default handling for this rule violation is error. You may ignore this error condition by issuing the command set_drc_handling D3 Warning.

When the handling is set to other than error, the tool will automatically make the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERS, test coverage may be reduced.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the master_observe argument and the desired time) and using the report_gates command for the gates in the SLAVE to MASTER path. This identifies the location of the blockage; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
N (G) not successfully observed by master_observe procedure. (D3-1)
```

N is the instance name of the MASTER memory element, and G is the gate ID number.

The summary message is:

```
N MASTERS not successfully observed by master_observe procedure. (D3)
```

N is the number of occurrences of rules violation D3.

D4

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

If you define the **skew_load** procedure, it must propagate the data value of the preceding scan cell (or scan chain input pin) to a MASTER memory element. The application performs this check using the simulated values of each time period of the **skew_load** procedure to trace back from a MASTER to its preceding scan cell output or scan chain input pin. The rule violation occurs if the **skew_load** procedure does not properly sensitize the path.

The default handling for this rule violation is error. Failure to satisfy this rule may result in inaccurate simulation results when you use the skew load option. The **skew_load** procedure is optional and you can avoid rules violations by removing the procedure definition.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the skew_load argument and the desired time) and using the report_gates command for the gates in the path. This identifies where the blockage occurred; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
Skew_load procedure not successful for MASTER %N (G). (D4-1)
```

N is the instance name of the MASTER memory element and G is the gate ID number.

The summary message is:

```
Skew_load procedure not successful for N MASTERS. (D4)
```

N is the number of occurrences of rules violation D4.

D5

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All memory elements (latches and flip-flops) must be scannable. The application performs this check after identifying all scan memory elements. The rule violation occurs for all memory elements not identified as part of a scan cell.

Note

 The D5 check simulates the circuit as purely combinational by using a sequential depth of 0 or 1 (as if the “set_pattern_type -Sequential” command is set to 0 or 1).

When the tool identifies a non-scan memory element, the tool classifies each element into one of the following types:

- INIT-0: If it is at 0 at the beginning of the first capture cycle.

- INIT-1: If it is at 1 at the beginning of the first capture cycle.
- INIT-X: If its state is unknown at the beginning of the first capture cycle and it may go to any state during capture.
- TIE-0: If it is always at 0 during capture.
- TIE-1: If it is always at 1 during capture.
- TIE-X: If it is always at an unknown state during capture.
- TLA: If it is always transparent when its clock is at its off state.

The model used (and reported in the D5 violation message) is determined from the element's value in the D5 simulation, at the end of load_unload and before entering capture. For example, if the element's value is 1 at that time, the tool models the element as a TIE-1. Note, however, that the tool simulates the TIE-0, TIE-1, and TIE-X models as INIT-0, INIT-1, and INIT-X models, respectively. This means, for example, that a non-scan TIE-0 model can be evaluated after the beginning of the first capture cycle and get a value other than 0.

Latches modeled as TIE-X gates become candidates for transparent latches, sequential transparent cells, or clocked sequential cells if you set the pattern type appropriately with the set_pattern_type command.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in some loss of test coverage.

The occurrence message is:

N (G) is a non-scan T1 converted to T2. (D5-1)

N is the instance name of the non-scan memory element, G is the gate ID number, T1 is the gate type (latch or flip-flop), and T2 is the gate type that models it (TIEX, TIE0, or TIE1).

The summary message is:

N non-scan memory elements converted to T gates. (D5)

N is the number of occurrences of rules violation D5, and T is the gate type that models the non-scan cell. The tool displays a summary message for each remodeled gate type.

D6

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All non-scan latches must behave as transparent latches. The application performs this check for all nonscan latches that are not set to a stable binary value. The rule violation occurs if a candidate latch fails one of the following conditions:

- If the latch creates a potential feedback path, that path must be broken by scan cells or non-scan cells other than transparent latches. For more information, refer to the `set_tla_loop_handling` command.
- The latch must have a propagable path to an observable point.
- The latch must be capable of passing a value when all defined clocks are at their off-state.
- All clock, set, and reset inputs of the latch must either be set to a determinate state when all clocks are off and pin constraints are set, or must not connect to defined clocks.
- The latch must not have more than one set/reset/clock input on when all defined clocks are at their off-state.

Failure to satisfy this rule can reduce test coverage. The default handling for this rule violation is warning. If you set the handling for this rule to ignore, the tool will not perform this check and the design's latches will not be checked for transparency.

For scan insertion in the dft -scan context, if you want the tool to consider non-transparent latches as scan candidates, you must turn test logic on (with the `set_test_logic` command) and do one of two things: 1) set the handling of D6 to ignore, in which case the tool does not perform the transparency check and automatically considers the non-scannable latches for scan insertion; or 2) use the `set_latch_handling` Scan command, in which case the tool performs the check and considers non-transparent latches for scan insertion.

The occurrence message is:

```
Latch N (G) not transparent due to R. (D6-1)
```

N is the instance name of the non-scan latch, G is the gate ID number, R is the reason it cannot be transparent, and D6 is the rule ID number.

The summary message is:

```
N latches not transparent due to R. (D6)
```

N is the number of occurrences of rules violation D6, and R is the reason. The application displays a summary message for each reason.

D7

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning
report_drc_rules: Supported

At the end of the **shift** procedure, the clock inputs of scan flip-flops must not be set to a one state. The application performs this check using the simulated values of the last time period of the **shift** procedure. The rule violation occurs if any clock input (not including set and reset lines) of any scan flip-flop (except COPY) is set to 1. A possible cause of a rules violation is an incorrect definition of the off-state of a clock.

Note

 Some design practices consider this condition acceptable.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in scan cells capturing data on the trailing edge of the capture clock pulse.

The occurrence message is:

```
Flip-flop N (G) has clock port set to stable high. (D7-1)
```

N is the instance name of the non-scan memory element, G is the gate ID number, and D7 is the rule ID number.

The summary message is:

```
N edge-triggered clock ports set to stable high. (D7)
```

N is the number of occurrences of rules violation D7.

D8

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

If a MASTER latch only propagates to a SLAVE and can only capture data when the SLAVE is inactive, a clock input of the MASTER latch must not be active when all clocks are off. The system uses the **master_observe** procedure to observe the values placed into the scan cell and no longer considers the SLAVE to be observable.

The application performs this check using the simulated values that result when all defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states.

The rule violation occurs if a clock input of a MASTER latch is not off, the MASTER latch only propagates to a SLAVE, and can only capture data when the SLAVE is inactive.

Note

 Some design practices consider this condition acceptable.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the clock input not held off, and by tracing back from this input, you can identify how to correct the problem.

The default handling for this rule violation is error because for certain rare LSSD scan cell designs it can result in a simulation mismatch during pattern verification in a timing-based simulator. Failure to satisfy this rule will result in data captured into the MASTER without the application of a capture clock.

Note

 If “set_drc_handling d8 warning” is in effect for DRC, the tool will automatically apply an SX cell constraint to each D8 failing master cell. This prevents the mismatches, but may reduce coverage. See the add_cell_constraints command description for information about the SX cell constraint.

If “set_drc_handling d8 note” is in effect for DRC, the tool will *not* apply any SX cell constraints for D8 violations, which ensures test coverage is not affected. However, you should use “note” handling only for cell architectures you know will not produce mismatches.

The occurrence message is:

```
MASTER latch N (G) allows data capture while clocks off. (D8-1)
```

N is the instance name of the non-scan memory element, and G is the gate ID number.

The summary message is:

```
Clocks at off-state allow data capture for N MASTER latches. (D8)
```

N is the number of occurrences of rules violation D8.

D9

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Seq_transparent procedures must not disturb scan cells, primary outputs, or previously calculated seq_transparent cells. The default handling for this rule violation is warning. The application performs this check during simulation of the **seq_transparent** procedure.

A rule violation occurs under any of the following conditions:

- If scan cell state elements change during the application of a **seq_transparent** procedure. Unless these scan state elements capture new data at the application of the capture clock, the system cannot observe them during patterns that apply the procedure.
- If disturbed scan cells supply the inputs of other scan cell state elements. The system cannot observe these other scan cell state elements during patterns that apply the procedure.
- If a primary output comes from disturbed circuitry, the system cannot observe these primary outputs during patterns that apply the procedure.
- If the application of the **seq_transparent** procedure disturbs a non-scan state element that previously captured an undisturbed value.

Failure to satisfy this rule results in restrictions on the use of these points for observation during simulation and test generation for patterns that apply the procedure. This can reduce test coverage.

If a disturbed cell must support a valid seq_transparent cell, the tool identifies the disturbed cell as a seq_transparent cell and issues a violation of type seq_transparent cell disturb. Otherwise, the tool will not identify it as a seq_transparent cell, and will instead issue a D9 violation of type unused seq_transparent cell disturb.

The occurrence message is:

T disturb occurred on N (G) in procedure P. (D9-1)

T is the type of disturb, N is the instance name of the disturbed gate, G is the gate ID number, P is the name of the **seq_transparent** procedure, and D9 is the rule ID number.

The summary message is:

N T disturbs occurred in procedure P. (D9)

N is the number of occurrences of a disturbance type, T is the type of disturbance, P is the name of the **seq_transparent** procedure, and D9 is the rule ID number. The tool issues a summary message for each disturbance type.

D10

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported for patterns -scan context only.

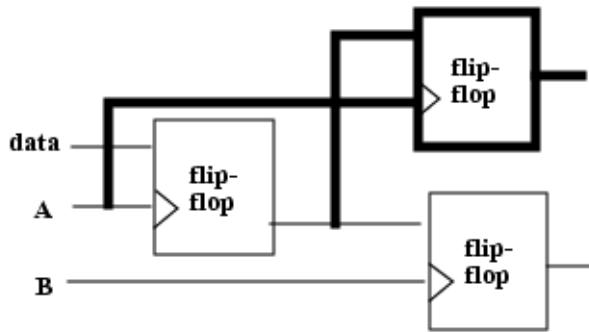
The transparent capture cells in **clock** procedures must not propagate both old and new data to other state elements. For example, a violation can occur when a clock procedure pulses two clocks, and a memory element clocked by the first applied clock feeds at least two other memory elements, clocked by each of the clocks.

To illustrate this, assume the clock procedure is as follows:

```
procedure clock clock_proc1 =
    force A 1 1;
    force A 0 2;
    force B 1 3;
    force B 0 4;
end;
```

Given this procedure, the highlighted flip-flop in the following figure, which gets old data from the first flip-flop when A pulses, violates this requirement.

Figure 7-24. Rule D10 Violation Example



The default handling for this rule violation is error. Failure to satisfy this rule will result in the source gate being modeled as a TIEX gate. You can suppress reporting the results of this check using `set_drc_handling`. However, regardless of how you set the handling, Tessent FastScan and Tessent TestKompress always perform this check on model violating gates with TIEX behavior.

The occurrence message is:

```
Cell N (G) has invalid transparency (X/Y) in procedure P. (D10-1)
```

N is the cell name, G is the gate ID number of the cell, X is the ID number of the gate capturing old data, Y is the ID number of the gate capturing new data, and P is the clock procedure name.

The summary message is:

```
There were N cells with invalid transparency. (D10)
```

N is the number of cells found to violate this rule.

D12

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In the state stability analysis stage, DRC simulates a number of shift cycles to calculate the stable value (load value) of nonscan memory elements at the beginning of capture.

If some nonscan memory elements need more than one shift cycle to be initialized, and its load value is observed into a scan cell, the parallel simulation test bench may have simulation mismatches. This is because in parallel simulation test bench the tool generally only has one parallel shift cycle, leading to insufficient shift cycles to initialize those nonscan elements.

If state stability analysis takes more than one cycle to calculate nonscan load value, then the tool issues the following warning message:

```
// Warning: DRC simulated N shift cycles to initialize some non-scan
// memory elements to their load values. (D12-1)
// Note: In order to avoid potential parallel simulation mismatches,
// M post shift cycles will be automatically added to the testbench as
// if the SIM_POST_SHIFT parameter keyword were set to M.
// This will slow down parallel simulation. To avoid slowing down
// parallel simulation, the number of cycles DRC simulates can be
// restricted using "set_stability_check -max_shift_cycles 1". However,
// initializing fewer non-scan memory elements may lead to lower test
// coverage.
```

N is number of shift cycles and M is the number of shifts required.

Pre-DFT Clock Rules (DFT_C Rules)

The tool uses the DFT_C rules to verify the clock specification and circuitry before DFT structures such as memory BIST, logic BIST, boundary scan, EDT, and OCC are inserted.

The tool invokes the DFT_C rules when the [check_design_rules](#) command is issued if the context has been specified as either dft -rtl or dft -no_rtl, with the [set_context](#) command, and the [set_dft_specification_requirements -memory_test](#) option and/or [-logic_test](#) options are set to “on”.

The following subsections describe these rules.

Simulation Contexts for Pre-DFT DRCs	2747
DFT_C1	2751
DFT_C2	2754
DFT_C3	2755
DFT_C4	2756
DFT_C5	2757
DFT_C6	2758
DFT_C7	2760
DFT_C8	2761
DFT_C9	2762
DFT_C10	2765
DFT_C11	2771
DFT_C12	2772
DFT_C13	2773

Simulation Contexts for Pre-DFT DRCs

This section describes the simulation contexts which are created by the DFT_C rules. This DRC always runs when transitioning from setup to analysis mode in the DFT context when no sub context is specified.

The created simulation contexts are used by the pre-DFT DRC rules described in the following sections. They can also be re-used by custom DRC rules you may have registered using the [register_drc](#) command.

Listed below are the simulation contexts created when transitioning from setup to analysis mode in the DFT context when no sub context is specified. You can obtain this list by using the

[get_simulation_context_list](#) command. To display those values in the GUI, you need to issue the following two commands:

```
set_gate_report simulation_context
set_current_simulation_context simulation_context_name
```

The *simulation_context_name* string is one of the names returned by the [get_simulation_context_list](#) command. When you have DRC violations for any of the DFT_C* types, the associated simulation context is set when you issue the “[analyze_drcViolation drc_name](#)” command or when you issue the “[set_gate_report drc_name](#)” command.

The simulation contexts are all derived from the stable_after_setup simulation context. This context is created by simulating the test_setup procedure followed by a state stability analysis using the specified input constraints. When ICL is elaborated, the reset and control signals of the IJTAG scan interface are auto constrained off to make sure the values loaded in the “scan resource instrument” side of the network are not disturbed by the state stability analysis. Refer to the [Sib wrapper](#) section for an explanation of the difference between the “scan resource instrument” and the “scan tested instrument” sides of the network.

When ICL is elaborated, the IJTAG network is always reset even if no [set_test_setup_icall](#) or [set_static_dft_signal_values](#) commands are issued. The DFT signals defined on the DataOutPorts of IJTAG nodes are forced directly on the pin of the associated instances because the pins need to be set to different values in different contexts. You typically only need to use the [set_test_setup_icall](#) or [set_static_dft_signal_values](#) commands when using “[set_dft_specification_requirements -logic_test on](#)” because you need to configure the child blocks so that sub_chain tracing can be done on them. When only using “[set_dft_specification_requirements -memory_test on](#)”, you typically do not need any test_setup iCall. The network is also typically not even created at that point. If the network is already created and you do need to use a test_setup iCall, you must request the IJTAG network to be synthesized if you are running in -rtl mode. The test_setup simulator only knows how to simulate gates. You will typically get E14 violations if you try to use an iCall in the test_setup and the IJTAG network is not synthesized. Use the following command to get the IJTAG network synthesized. You do note need these commands when “[set_dft_specification_requirements -logic_test on](#)” because it already causes the entire design to be synthesized.

```
set_attribute_value -name synthesize_before_analysis \
[get_modules -of_instances [get_ijtag_instances]]
```

In the following section, the DFT signals are mentioned and their attributes are referenced. For a description of their usage, [reset_value](#), [default_value_in_all_test](#) and [value_in_pre_scan_drc](#) property, refer to the description of the [add_dft_signals](#) and the [register_static_dft_signal_names](#) commands.

Simulation Contexts

- [dft_glb](#)

This simulation context is created on top of the stable _after_ setup simulation context. The DFT signals with usage “global_dft_control” are forced to their “default_value_in_all_test” value. The targets of the [add_dft_control_points](#) command using a DFT signal with usage “global_dft_control” are also forced taken into account the -inverse_dft_signal_source switch. All clocks (source, generated and branch) are forced to X. This allows providing better isolation of the violations. For example, if the source of a branch clock is driven by a constant, it will be reported once by [DFT_C2](#), yet [DFT_C1](#) and [DFT_C6](#) will still allow checking that the memories and the scan elements are properly driven by the branch clock.

- [dft_glb_fe](#)

This simulation context is created on top of the [dft_glb](#) simulation context. All func_en pins of clock gaters are forced to their active value. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “[add_dft_clock_enables](#)-usage func_en” command. The tool only forces the clock gaters in the combinational fanin of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the [add_dft_clock_enables](#) command to point directly in the func_en pin you want asserted anyway.

- [dft_glb_ns](#)

This simulation context is created on top of the [dft_glb](#) simulation context. The “nonscan_test” DFT signal is forced to its active value. The DFT signals with usage “logic_test_control” are forced to their reset value. The “scan_en” and the “edt_update” DFT signals are also forced to their inactive value. The target of the [add_dft_control_points](#) command using any of those DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

- [dft_glb_ns_fe](#)

This simulation context is created on top of the [dft_glb_ns](#) simulation context. All func_en pins of clock gaters are forced to their active value. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “[add_dft_clock_enables](#)-usage func_en” command. The tool only forces the clock gaters in the combinational fanin of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the [add_dft_clock_enables](#) command to point directly in the func_en pin you want asserted.

- [dft_glb_ns_fe_tck](#)

This simulation context is created on top of the [dft_glb_ns_fe](#) simulation context. The tck_select DFT signal is forced to its active value, justifying TCK onto the functional clock trees.

- [dft_glb_ns_fe_x](#)

This simulation context is created on top of the [dft_glb_ns](#) simulation context. All func_en pins of clock gaters are forced to X. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “[add_dft_clock_enables -usage func_en](#)” command. The tool only forces the clock gaters in the combinational fanin of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the [add_dft_clock_enables](#) command to point directly in the func_en pin you want asserted.

- [dft_glb_s](#)

This simulation context is created on top of the [dft_glb](#) simulation context. All DFT signals with usage “logic_test_control” are forced to their “value_during_pre_scan_drc” value. All DFT signal with usage “scan_mode” are forced to X. The DFT signal “edt_update” is forced to 0.

- [dft_glb_s_se_0](#)

This simulation context is created on top of the [dft_glb_s](#) simulation context. The “scan_en” DFT signal is forced to 0. The target of the [add_dft_control_points](#) command using the “scan_en” DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

- [dft_glb_s_se_0_te_1](#)

This simulation context is created on top of the [dft_glb_s_se_0](#) simulation context. All test_en pins of clock gaters are forced to their active value. The test_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “[add_dft_clock_enables -usage test_en](#)” command. Only the clock gaters that fanout to at least one DFF primitive with the is_non_scannable attribute equal to false are forced. The others clock gaters are not forced so as not to disturbed test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the [add_dft_clock_enables](#) command to point directly in the test_en pin you want asserted.

- [dft_glb_s_se_0_te_x](#)

This simulation context is created on top of the [dft_glb_s_se_0](#) simulation context. All test_en pins of clock gaters are forced to X. The test_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “[add_dft_clock_enables](#) -usage test_en” command. Only the clock gaters that fanout to at least one DFF primitive with the is_non_scannable attribute equal to false are forced. The others clock gaters are not forced so as not to disturbed test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the [add_dft_clock_enables](#) command to point directly in the test_en pin you want asserted.

- [dft_glb_s_se_x](#)

This simulation context is created on top of the [dft_glb_s](#) simulation context. The “scan_en” DFT signal is forced to X. The target of the [add_dft_control_points](#) command using the scan_en DFT signals are also forced to X.

- [dft_glb_s_se_1](#)

This simulation context is created on top of the [dft_glb_s](#) simulation context. The “scan_en” DFT signal is forced to 1. The target of the [add_dft_control_points](#) command using the scan_en DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

DFT_C1

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each memory clock port has a defined clock in its controlling fanin. The default handling of this rule violation is error and cannot be changed.

This rule check is executed only for designs that have memory instances, as reported by the [report_memory_instances](#) command or when the [set_dft_specification_requirements](#) -memory_test command option are set to “on”. It is not executed in any other case or context. This DRC rule is performed using the [dft_glb_ns_fe](#) simulation context.

This rule check ensures that each clock pin on a memory instance is driven by a clock source that was previously defined with the [add_clocks](#) command. The check is performed with a backward trace from each memory instance clock pin. A rule violation occurs if the backward trace stops at a port or at a pin that has not been declared as a clock using the [add_clocks](#)

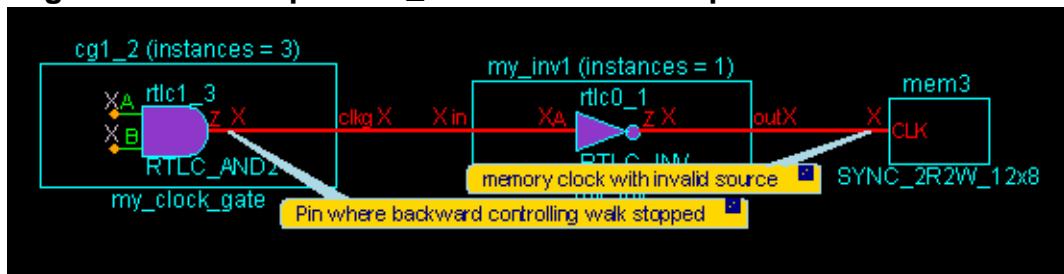
command. The tool will then report this port or pin in an error message similar to the one shown here:

```
// Error: The memory clock pin 'blockA_I1/mem1/CLKW' (23.28) source
// tracing stopped at pin 'pll_Inst/pll_out_2' (13.0)
// Correct the block conditions or use the 'add_clocks -period' if this
// is an embedded oscillator or 'add_clocks -reference' if this is an active
// PLL or clock divider. (DFT_C1-1)
```

This type of trace violation can have many different reasons. You should check your test_setup procedure and/or design constraints and ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design's clock source. If the blockage is caused by a clock gating cell, you can use the [add_dft_clock_enables](#) command to point to the functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you use the [add_dft_control_points](#) command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path; this is shown in [Figure 7-25](#) where a PLL instance is black boxed. In this type of case, you can use the -reference or -reference_inv option of the [add_clocks](#) command to bridge the black box tracing gap. Assuming that this black box was an internal clock oscillator, you can use the [add_clocks -period](#) option to generate an internal clock source (for example, `add_clocks pll_Inst/pll_out_2 -period 10ns`). If you have an associated ICL module for the PLL or embedded oscillator module, the [add_clocks](#) command will automatically be inferred as described in the Description section of the [add_clocks](#) command.

Figure 7-25. Example DFT_C1 violation in the presence of a blackbox



The error messages generated by those checks are listed in [Table 7-4](#).

Table 7-4. Error Messages Generated by the Clock Tracing Checks

Type 1: The <clock> source tracing stopped at <location>. Correct the blocking conditions or use the 'add_clocks -period' if this is an embedded oscillator or 'add_clocks -reference' if this is an active PLL or clock divider.
Type 2: The <clock> is sourced by port <post> but its period was not defined using the 'add_clocks -period' command.
Type 3: The <clock> is sourced by a constant value of '0'.
Type 4: The <clock> has no source.

Table 7-4. Error Messages Generated by the Clock Tracing Checks (cont.)

Type 5: The memory <clock> source tracing stopped at <location> and this node was the target of an add_dft_clock_mux. An add_dft_clock_mux node can only be behind a clock, and cannot feed directly to the memory clocks.

If a clock becomes differential, the differential buffer must be modeled with a WIRE gate as shown below:

```
model diff_buf (out_p, out_n, in_p, in_n) (
    input (in_p, in_n) (
        output (out_p, out_n) (
            primitive = _inv (in_n, in_n_inv);
            primitive = _buf (in_p, in_p_buf);
            primitive = _wire (in_p_buf, in_n_inv, out_p);
            primitive = _inv (out_p, out_n);
        )
    )
)
```

[Figure 7-27](#) on page 2754 shows the schematic view of a differential clock path as well as the flat model representation. When a backward trace reaches a WIRE gate, a set of checks is performed to make sure the differential clock path is correct. The error messages generated by those checks are listed in [Table 7-5](#) on page 2753.

Table 7-5. Error Messages Generated by Differential Clock Checks

Type 1: The <clock> became differential but failed a rule along the differential clock path. A side of the differential pair traced to an add_dft_clock_mux location <location> which is illegal for a differential signal.
Type 2: The <clock> became differential but only one side of the differential source traced to a defined clock source. When one element of a differential clock reaches a defined clock, the other must too.
Type 3: The <clock> became differential but failed a rule along the differential clock path. A side of the differential pair traced to a constant value.
Type 4: The <clock> became differential but it reached a source on one of the differential sides that is not a declared clock nor a WIRE gate used to model a differential buffer.
Type 5: The <clock> became differential but reached two different WIRE gates. A differential buffer must be modeled with a single 2 input WIRE gate.
Type 6: The <clock> became differential but the inversion from the two inputs of one WIRE gate was different when reaching the other WIRE gate. A Differential clock buffer is modeled with a WIRE gate and there must be no relative inversion between the two input of the WIRE gate and the WIRE gate sourcing them.
Type 7: The memories on clock domain <domain_label> are sourced by two different differential clock buffers: <differential buffer 1> <differential buffer 2> You must separate them into two clock domains using 'add_clock <clock> -branch' on at least one of them.

Table 7-5. Error Messages Generated by Differential Clock Checks (cont.)

Type 8: The differential clock <p_side> and <n_side> reached a WIRE gate but the inversion from both of them was not different when reaching that WIRE gate.

Figure 7-26 shows example of a violation that is generated when an inverter is added on one side of the differential clock path and not on the other.

Figure 7-26. Example DFT_C1 Violation on Differential Clocks

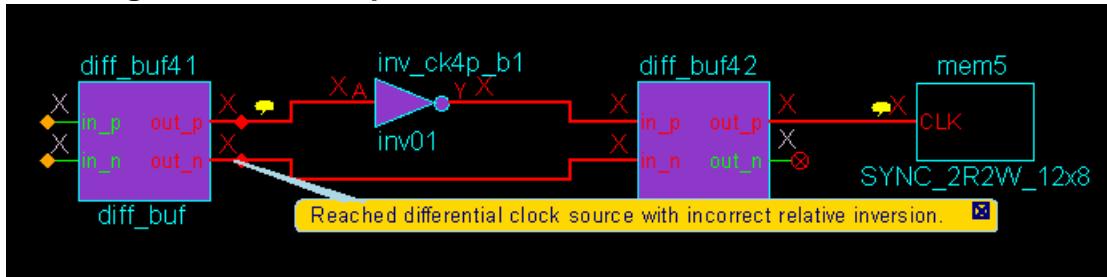
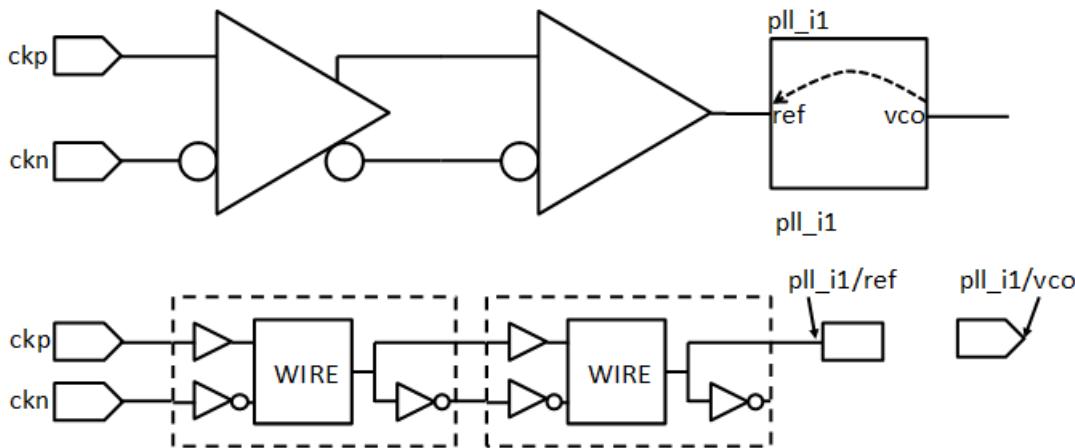


Figure 7-27. Schematic and Flat Model View of a Differential Clock Path



DFT_C2

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each clock of type “branch” has a defined clock in its controlling fanin. The default handling of this rule violation is error and cannot be changed.

This rule check is executed only for designs that have designed branch clocks (as defined using the [add_clocks](#) -branch command) and when the [-memory_test](#) and/or [-logic_test](#) option of the [set_dft_specification_requirements](#) command is set to “on”. It is not executed in any other case

or context. It is not executed in any other case or context. This DRC rule is performed using the [dft_glb_fe](#) simulation context.

This rule check ensures that each identified clock of type branch (as defined by the [add_clocks](#)-branch option) is driven by a clock source defined using the [add_clocks](#) command. The check is performed using a backward trace from each clock branch. A rule violation occurs if the backward trace ends at a port or instance pin that has not been declared as a clock using the [add_clocks](#) command as shown in [Figure 7-26](#). The tool reports the port or pin where the traced stopped as shown in the error message below:

```
// Error: The clock branch 'clk_B_buf_2/Y' (1706.0) is sourced by
// 'clk_B' (2.0) but the source was not declared as a clock using the
// 'add_clocks' command. (DFT_C2-1)
```

This type of a trace violation can have many different reasons. You should check your test_setup procedure and/or design constraints to ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design's clock source. If the blockage is caused by a clock gating cell, you can use the [add_dft_clock_enables](#) command to point to its functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you can use the [add_dft_control_points](#) command. Finally, to inject an alternative source to a clock net, you use the [add_dft_clock_mux](#) command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path. In this case, you can use the -reference or -reference_inv option of the [add_clocks](#) command to bridge the black box tracing gap. Assuming that the black box was an internal clock oscillator, you can use the [add_clocks](#)-period option to generate an internal clock source (for example, [add_clocks pll_Inst/pll_out_2 -period 10ns](#)). If you have an associated ICL module for the PLL or embedded oscillator modules, the [add_clocks](#) command will automatically be inferred as described in the Description section of the [add_clocks](#) command.

See [Figure 7-25](#) and [Figure 7-26](#) in the [DFT_C1](#) section for examples showing how the violations are displayed in the schematic. The error messages generated when checking the rules are listed in [Table 7-4](#) and [Table 7-5](#). The bottom error message in [Table 7-4](#) and the bottom two error messages in [Table 7-5](#) are only relevant to [DFT_C1](#) and do not apply to [DFT_C2](#).

DFT_C3

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Vерifies that each generated clock has a defined clock in the controlling fanin of its reference. The default handling of this rule violation is error and cannot be changed. For a description of what constitutes a generated clock, see the description of the `add_clocks` command.

This rule check is executed only for designs that have defined, generated clocks (as defined using the `add_clocks`-reference command) and when the -memory_test and/or the -logic_test options of the `set_dft_specification_requirements` command is set to "on". It is not executed in any other case or context. This DRC rule is performed using the `dft_glb_fe` simulation context.

The check is performed using a backward trace from the specified reference and optional reference inverse pin. A rule violation occurs if the backward trace ends at a port or a pin that has not been declared as a clock using the `add_clocks` command. The tool reports the port or pin where the trace stopped in an error message similar to the one shown here:

```
// Error: The reference pin 'clk_REF' (3.0) of the derived clock source
// at 'pll_Inst/pll_out_2' (5.0) is sourced by 'clk_REF' (3.0)
// but the source was not declared as a clock using the 'add_clocks'
// command. (DFT_C3-1)
```

This type of a trace violation can have many different reasons. You should check your test_setup procedure and design constraints to ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design's clock source. If the blockage is caused by a clock gating cell, you can use the `add_dft_clock_enables` command to point to its functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you can use the `add_dft_control_points` command. Finally, to inject an alternative source to a clock net, you use the `add_dft_clock_mux` command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path. In this case, you can use the -reference or -reference_inv option of the `add_clocks` command to bridge the black box tracing gap. Assuming that the black box was an internal clock oscillator, you can use the `add_clocks`-period option to generate an internal clock source (for example, `add_clocks pll_Inst/pll_out_2 -period 10ns`). If you have an associated ICL module for the PLL or embedded oscillator modules, the `add_clocks` command will automatically be inferred as described in the Description section of the `add_clocks` command.

See [Figure 7-25](#) and [Figure 7-26](#) in the [DFT_C1](#) section for examples showing how the violations are displayed in the schematic. The error messages generated when checking the rules are listed in [Table 7-4](#) and [Table 7-5](#). The bottom error message in [Table 7-4](#) and the bottom two error messages in [Table 7-5](#) are only relevant to [DFT_C1](#) and do not apply to [DFT_C3](#).

DFT_C4

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that a clock of type “generated” or “branch” is not itself in the fanin of its reference source. A generated clock is a clock that was defined with the add_clocks -reference option specified. The default handling of this rule violation is error and cannot be changed.

This DRC rule is performed using the [dft_glb_ns_fe](#) simulation context. This rule check is executed only for designs that have declared branch clocks (as defined using the [add_clocks -branch](#) command) or generated clocks (as defined using the [add_clocks -reference](#) command) and when the [-memory_test](#) option of the [set_dft_specification_requirements](#) command is set to “on”. It is not executed in any other case or context.

This rule check ensures that the reference of each generated clock source traces to a source that is not affected by itself. A rule violation occurs if a loop is detected in the fanin of the referenced pin. The tool reports the violation as shown below. The schematic that is shown you issue the [analyze_drcViolation](#) command highlights the circular path:

```
// Error: The reference pin 'clk_pll_buf/Y' (1683.0) of the derived clock
// source at 'pll_Inst/pll_out_3' (5.0)
// is driven by itself via the clock source. (DFT_C4-1)
```

Figure 7-28. Example of DFT_C4 Violation



DFT_C5

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each clock source has a specified period set using the “add_clocks -period” command. The default handling of this rule violation is error and cannot be changed.

This rule check is executed only for designs that have memory instances or declared generated or branch clocks and when the [-memory_test](#) option of the [set_dft_specification_requirements](#) is set to “on”. It is not executed in any other case or context. This rule does not use a simulation context. It simply verifies the source of the clock extracted by [DFT_C1](#), [DFT_C2](#), and [DFT_C3](#) have a specified period so that the SDC clocks with a known period can be created later in the flow when running the [extract_sdc](#) command. The clock periods are also used by the

[create_patterns_specification](#) when creating the signoff and manufacturing patterns specification wrappers.

This rule check ensures that each clock has a well-defined period.

A rule violation occurs if a clock source is found that does not have a well-defined period, as shown in this example:

```
add_clocks clk_REF
# add_clocks clk_REF -period 100ns
add_clocks clk_A -pulse_always -period 10ns
add_clocks clk_B -period 15ns

add_clocks pll_Inst/pll_out_1 -reference_pin clk_REF
add_clocks pll_Inst/pll_out_2 -reference_pin clk_REF

check_design_rules
```

Notice that the clock port “clk_REF” does not have a defined period. However, it is used as the reference of the generated clocks pll_Inst/pll_out_1 and pll_Inst/pll_out_2. The tool reports a violation for the above example as shown here:

```
// Error: The clock 'clk_REF' must be defined with a period.
//         The clock is used by memory_bist. (DFT_C5-1)
```

DFT_C6

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each scannable flip-flop clock port has a defined clock in its controlling fanin. The default handling of this rule violation is error and cannot be changed.

This rule check is executed when the [set_dft_specification_requirements -logic_test](#) command option is set to “on”. It is not executed in any other case or context. This DRC rule is performed using the [dft_glb_s_se_0_te_1](#) simulation context.

This rule check ensures that each clock pin on a scannable flip-flop is driven by a clock source that was previously defined with the [add_clocks](#) command. The check is performed with a backward trace from each scannable flip-flop instance. The clock gaters which becomes open because of the [dft_glb_s_se_0_te_1](#) simulation context are verified for proper operation in [DFT_C8](#). A rule violation occurs if the backward trace stops at a port or at a pin that has not

been declared as a clock using the [add_clocks](#) command. The tool will then report this port or pin in an error message similar to the one shown here:

```
// Error: The clock pin 'blockA_I1/mem1/CLKW' (23.28) source
// tracing stopped at pin 'pll_Inst/pll_out_2' (13.0)
// Correct the block conditions or use the 'add_clocks -period' if this
// is an embedded oscillator or 'add_clocks -reference' if this an active
// PLL or clock divider. (DFT_C6-1)
```

This type of trace violation can have many different reasons. You should check your test_setup procedure and/or design constraints and ensure that they are properly sensitizing a controlling backward path from the scannable flip-flop instance clock pin to the design's clock source. If the blockage is caused by a clock gating cell, you can use the “[add_dft_clock_enables](#) -usage test_en” command to point to the test_en enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you use the [add_dft_control_points](#) command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path where a PLL instance is black boxed. In this type of case, you can use the -reference and -reference_inv option of the [add_clocks](#) command to bridge the black box tracing gap. Assuming that this black box was an internal clock oscillator, you can use the [add_clocks](#) -period option to generate an internal clock source (for example, add_clocks pll_Inst/pll_out_2 -period 10ns). If you have an associated ICL module for the PLL or embedded oscillator module, the [add_clocks](#) command will automatically be inferred as described in the description section of the [add_clocks](#) command. See the description of the [ProcedureStep](#) wrapper for an example of such ICL module.

The error messages generated by those checks are listed in [Table 7-6](#). [Figure 7-29](#) below is an example of the schematic shown when the [analyze_drcViolation](#) command is issued on a Type 1 violation.

Figure 7-29. Example DFT_C6 Violation

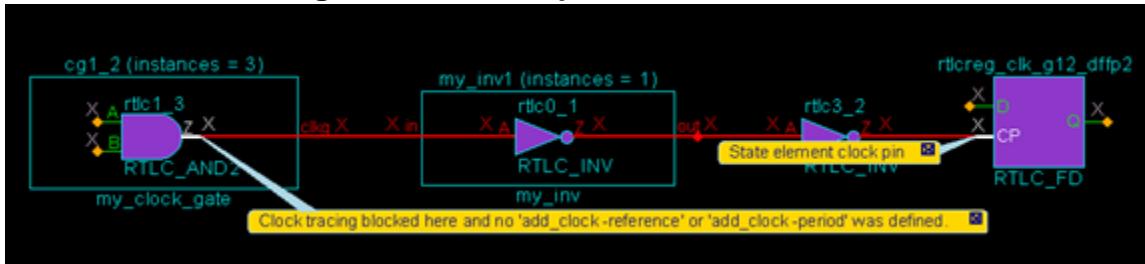


Table 7-6. Error Messages Generated by the Clock Tracing Checks

Type 1: The <clock> source tracing stopped at <location>. Correct the blocking conditions or use the “ add_clocks -period ” command if this is an embedded oscillator or “ add_clocks -reference ” if this is an active PLL or clock divider.

Type 2: The <clock> is sourced by port <post> but its period was not defined using the “ add_clocks -period ” command.
--

Table 7-6. Error Messages Generated by the Clock Tracing Checks (cont.)

Type 3: The <clock> is sourced by a constant value of “0”.
Type 4: The <clock> has no source.
Type 5: The scannable flip-flop <clock> source tracing stopped at <location> and this node was the target of an add_dft_clock_mux. An add_dft_clock_mux node can only be behind a clock, and cannot feed directly to the memory clocks.

For differential paths, the same handling of different clocks shown in the [DFT_C1](#) description applies to DFT_C6 except that the start points are clock pins of scannable flip-flops instead of memory clocks.

DFT_C7

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that the clock gating cells in the fanin of memory clocks are properly implemented.

This rule check is executed only for designs that have memory instances or declared generated or branch clocks and when the -memory_test option of the [set_dft_specification_requirements](#) is set to “on”. It is not executed in any other case or context. This DRC rule is performed using the [dft_glb_s](#) simulation context. Any gate that was marked between a clock and a memory clock pin, a branch clock or the reference of a generated clock which becomes blocking when the simulation context is switch from [dft_glb_ns_fe](#) to [dft_glb_ns](#) is checked in this DRC rule.

Several checks are performed to verify that the clock gating meets the specific checks to make sure that they will remain compatible to logictest. You can use the [add_dft_control_points](#) command on the func_en pin to disable the clock gating completely. If you fail one of those violations, however, those clock gaters will remain untestable during logictest. Only clock gaters disabled by a specified or inferred [add_dft_clock_enables](#) signal are verified by this DRC. See [report_dft_clock_enables](#) for more information. The error messages generated by those checks are listed in [Table 7-7](#). clock gaters feeding branch clocks or the reference of generated clocks are disabled with the DFT signal “all_test” because they need to be disable in both the non-scan and scan based test modes.

Table 7-7. Error messages Generated by DFT_C7 Checks

A unique dft_clock_enable signal of type func_en could not be found in the fanin of <node>. The multiple dft_clock_enable signals of type func_en found in the fanins are: <node1> <node2> ...
The clock gating cell <instance> is a primitive cell of type <type>. A clock gating cell must be a primitive of type AND or OR.

Table 7-7. Error messages Generated by DFT_C7 Checks (cont.)

The clock gating cell 'clock_gate_and_bad1/and3/Y' is a primitive cell of type 'AND (OR)' but it has 3 inputs. Only 2 input primitives are allowed as clock gating cells.
The clock gating cell <node> has no latch in its enable path.
The clock gating cell <node> is an AND gate but the latch in its enable path is active high instead of active low.
The clock gating cell <node> has a latch in its enable path but its clock input is not connected to the same driver that feeds the clock input of the clock gating cell.
The clock gating cell <node> is an OR gate but the latch in its enable path is active low instead of active high.

DFT_C8

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that the clock gating cells in the fanin of scannable flip-flops are properly implemented. The default handling of this rule violation is error but it can be downgraded to warning.

This DRC rule is performed using the [dft_glb_s_se_0_te_x](#) simulation context. Any gate which was marked between a clock and a scan element which becomes blocking when the simulation context is switch from [dft_glb_s_se_0_te_1](#) to [dft_glb_s_se_0_te_x](#) is checked in this DRC rule.

This rule check is executed only for designs that have scannable flip-flops and when the [-logic_test](#) option of the [set_dft_specification_requirements](#) is set to “on”. It is not executed in any other case or context.

Several checks are performed to verify that the clock gating meets the specific checks to make sure that they will remain compatible to logictest. If you fail one of those violations, you can permanently disable the clock gater using “[add_dft_control_points -dft_signal ltest_en](#)” on the test_en pin of the clock gater. However, the fanin of the func_en pin of the clock gater will be untestable during logictest. Only clock gaters disabled by a specified or inferred [add_dft_clock_enables](#) signal are verified by this DRC. The error messages generated by those checks are listed in [Table 7-8](#). Note that those checks are run only if the clock gater is not a cell with the [simulation_function](#) attribute equal to “clock_gating_and” or “clock_gating_or”. Those clock gaters already go through a formal verification when the cell library is loaded and don’t need to be checked again. The formal check of the library parser is more flexible and allows NAND and NOR gates.

Table 7-8. Error messages Generated by DFT_C8 Checks

The clock gating cell <instance> is a primitive cell of type <type>. A clock gating cell must be a primitive of type AND or OR.
The clock gating cell 'clock_gate_and_bad1/and3/Y' is a primitive cell of type 'AND (OR)' but it has 3 inputs. Only 2 input primitives are allowed as clock gating cells.
The clock gating cell <node> has no latch in its enable path.
The clock gating cell <node> is an AND gate but the latch in its enable path is active high instead of active low.
The clock gating cell <node> has a latch in its enable path but its clock input is not connected to the same driver that feeds the clock input of the clock gating cell.
The clock gating cell <node> is an OR gate but the latch in its enable path is active low instead of active high.

DFT_C9

Category: Asynchronous Set Reset

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that it is possible to disable the asynchronous set or reset pin of every scannable flip-flop. The default handling of this rule violation is error but it can be downgraded to ignore.

This rule check is executed only for designs that have scannable flip-flops, and when the -logic_test option of the [set_dft_specification_requirements](#) is set to “on”. It is not executed in any other case or context. This DRC rule is performed using the [dft_glb_s_se_1](#) simulation context.

Three types of errors can be generated by the DFT_C9 DRC, and they are all shown in [Table 7-9](#). The situation detected by the first two errors are not auto fixed and must be corrected explicitly. The situation detected by the last error is auto fixed if “[set_drc_handling](#) DFT_C9 -auto_fix” was specified or left to default to on.

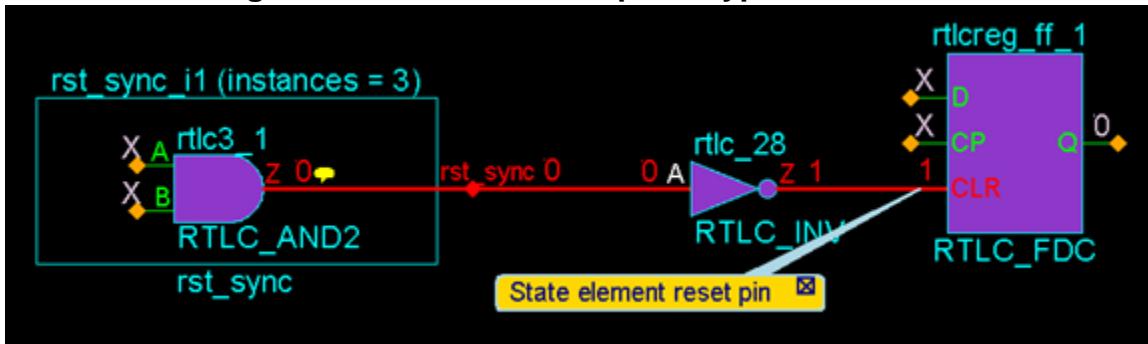
Table 7-9. Error messages Generated by DFT_C9 Checks

Type 1: When the async_set_reset source <node_name> is asserted '0' '1' <num1> set reset pins on DFFs are forced off but <num2> set reset pins on DFF are forced on. Use another 'add_dft_control_point' command to break the path to those flops or declare them as non-scannable using 'add_nonscan_instance' command. The list of set pins on DFF which are activated when the source is asserted '0' '1' are: <pin1> <pin2> ...
Type 2: The active high low set reset pin <gate_pin1> is sourced by <gate_pin2> and it has a constant simulated value of '1' '0' which forces the set reset pin active.

Table 7-9. Error messages Generated by DFT_C9 Checks (cont.)

Type 3: The active high|low set|reset pin <gate_pin1> source tracing stopped at pin <gate_pin2>. Correct the blocking conditions or use 'add_dft_control_point -type async_set_reset' to make it controllable during logic test

The first type of error reports nodes that source asynchronous set and reset of scannable flip-flops having different active polarity. You have two options to resolve this issue. Solution 1 is to use the “[add_dft_control_points nodes](#) -type [async_set_reset](#)” command to specify additional locations so that the active polarity of the asynchronous set and reset of all scannable flip-flop in the fanout of the specified **nodes** is the same. The other option is to declare the group of flip-flops listed in the error message as non-scannable using the [add_nonscan_instances](#) command. [Figure 7-30](#) is an example of the schematic shown when using the [analyze_drcViolation](#) command on a violation of type 1. The path between the source and one of the flops that has its set or reset pin forced on is shown. The source node is simulated to the value that makes the set or reset pins on on the minority group of scannable flip-flops.

Figure 7-30. DFT_C9 example of type 1 violation


The second type of error reports nodes that sources asynchronous set and reset pins of scannable flip-flops which simulate to a constant value and the constant value activates the set and reset pins of scannable flip-flops. To correct this situation, you must verify your [test_setup](#) procedure or your specified [add_dft_control_points](#) specification to understand if they not responsible for those scannable flip-flops to be held in reset or set. If this is truly the functional intent, declare those flip-flops as non-scannable using the [add_nonscan_instances](#) command.

The third type of error will not happen if you disable auto fixing by issuing the following command:

```
set_drc_handling DFT_C9 -auto_fix off
```

The tool traces the controlling fanin of the set/reset pins and issues the “[add_dft_control_points](#) -type [async_set_reset](#)” command and switch at the node where the trace stopped. The tool issues a summary message similar to the following:

```
// Note: There were 9 'add_dft_control_points -type async_set_reset'
// commands inferred from DRC. Use report_dft_control_points to see them.
```

Note

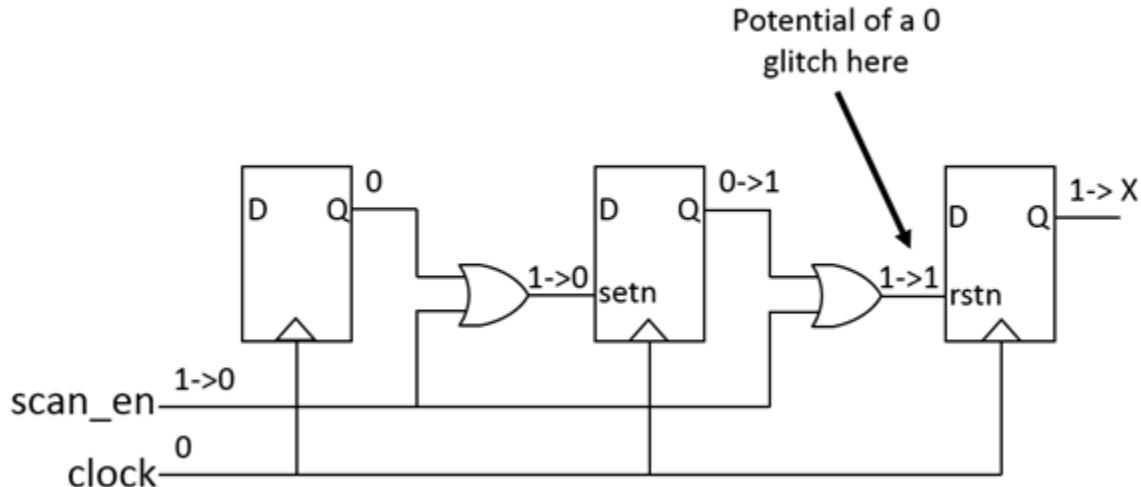
If the auto connection is inferred on a port of the current design and the port connects to the pad_io function of a pad cell, the DFT connection will be inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it will be connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it will connect to the from_sji_mux function when it exists, otherwise it will connect to the from_pad function.

If you are running on a gate-level netlist, it is possible that you will have more inferred DFT control than was actually needed. This happens when the synthesis tools perform complex restructuring and introduced X re-convergence logic in the set and reset logic paths. This is rare in RTL because the quick synthesis tool does not do any restructuring. If it happens in the RTL, then it means you have X re-convergence logic coded in your RTL. To remove this pessimism, explicitly specify the “[add_dft_control_points](#)-type `async_set_reset`” commands on the nodes where you want them to be, and the nodes will be simulated to determine their effects. If the DRC is able to force the specified node to a value to make all the asynchronous set and reset of the scannable flop-flop in its fanout inactive, then the presence of X re-convergence path will not matter.

If you have set or reset signals going to flip-flops which themselves are the source of the set or reset signals of other flip-flops, the tool performs an extra check to verify that no race condition can occur in the scan test when the `scan_en` signal is toggled from 1 to 0.

As shown in [Figure 7-31](#), the race condition happens if the reset/set value of the flip-flops feeding the set or reset signals of the other flip-flops is the inactive value for the set or reset signals of the other flip-flops. When `scan_en` goes low, the gating of the resets is removed. If the source flop was scan loaded with the active value for the set or reset signals of the flip-flops it drives and its reset becomes active when `scan_en` goes low, the flip-flops in the fanout may or may not be reset.

Figure 7-31. Race Condition Check



If the source flop had time to reset to the inactive value before the scan_en released the set/reset signal of the flip-flops in its fanout, the flip-flops in the fanout may not be reset or set.

Conversely, if the source flop did not have time to reset to its inactive value before the scan_en released the set/reset signal of the flip-flops in its fanout, the flip-flops in the fanout will be reset or set.

ATPG will most often recognize this situation and X out the state of the destination flip-flops, which causes lots of masking. To prevent the race, the set or reset signal of the source or destination flip-flops is disabled with the ltest_en signal instead of the scan_en signal based on which one has less affected logic that can no longer be tested. When this happens, the tool issues the following note:

```
// Note: The DFT_C9 DRC found the following node sourcing set reset pins
//        of scannable flip-flops:
//        rst
//        It is disabled with 'ltest_en' because it is involved in a
//        cascaded set/reset configuration that generates glitches during
//        scan test.
```

DFT_C10

Category: Clock

Context Supported: dft

Default Handling: Error/Warning

report_drc_rules: Supported

Verifies that an injection node for an OCC or TCK MUX can properly control a clock source. The default handling of the rule violation is a warning for the logic test DFT specification, and an error for the -dft etchecker sub context.

This rule check is executed with severity warning if you issue the [set_dft_specification_requirements](#) command with the -logic_test switch as follows:

```
set_dft_specification_requirements -logic_test on
```

It is executed with a severity of error in the context dft -etchecker.

The validated DFT injection node can then be used using the “get_clock_option -dft_inject_node” command and switch to determine where to inject the TCK multiplexer—see the [-tck_injection](#) switch description to the [set_dft_specification_requirements](#) command.

You can upgrade or downgrade this DRC to error, warning, note, or ignore using the [set_drc_handling](#) command.

You specify the DFT injection node when you create a new clock using the “add_clocks -dft_inject_node” command and switch. To specify the injection node for an existing clock, use the “set_clock_options -dft_inject_node” command and switch. This DRC rule is performed using the [dft_glb_s_se_0_te_1](#) simulation context.

The DFT_C10 runs to ensure that the injected DFT node meets the following requirements:

- Only one injection node should be defined for a clock source. The clock source can be an ordinary clock source (a primary pin) or a derived clock source (for example, a PLL output).
- An injection node does fanout to scan elements and memories as the corresponding clock source.
- The DFT inject node is in the controlling fanout of a defined clock source. In case of differential inputs, the DFT inject node is the node or is in the controlling fanout of the node where the clock became single ended.
- All scan elements sourced by clock source are in the fanout of the DFT inject node.
- There are no references of generated clock in the fanout of the DFT inject node.
- All branch clocks sourced by a clock source are in the fanout of the DFT inject node.
- The injection node is not specified where a constant setting is propagated or the node is open.
- An injection node has to be specified in case of a clock source that is driving a generated clock source reference and scan elements/memories.
- The path between the clock source and the injection node must be transparent to ensure that the above rules can be fully checked.

Table 7-10. Error and Warning Messages Generated by DFT_C10

Warning/Error: The backward tracing from the specified -dft_inject_node '<injection_node>' did not reach the clock '<defined_clock_source>'. The tracing stopped at node '<tracing_end_node>'. [<stop_location_reason_addendum>] (DFT_C10-x).
Warning/Error: The backward tracing from the specified -dft_inject_node '<injection_node>' which is where the clock '<defined_clock_source>' becomes single ended at node <differential_output_node>. The tracing stopped at node '<tracing_end_node>'. (DFT_C10-x).
Warning/Error: Clock '<clock_source>' fanouts to ['<number>' scannable element[s]] [and] ['<number>' BIST'ed memory cell inputs[s]] which [is][are] not in the fanout of the specified -dft_inject_node '<injection_node>', which are 'failing_element_1' 'failing_element_2' ... 'failing_element_10' (DFT_C10-x)
Warning/Error: The clock source '<clock_source>' is the source of the reference of the generated clock '<gen_clock_source>' and require the specification of a correct injection node. Use 'set_clock_option clk -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and all memories sourced by '<clock_source>' are in the fanout of the DFT inject node. 3) There are no references of generated clock in the fanout of the DFT inject node. 4) There are no branch clocks in the fanout of the DFT inject node. (DFT_C10-x)

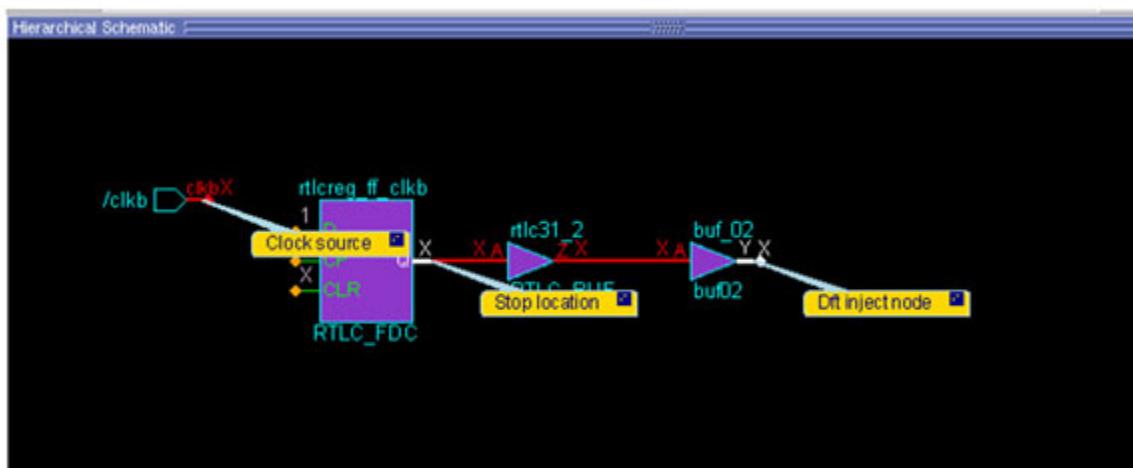
Table 7-10. Error and Warning Messages Generated by DFT_C10 (cont.)

Warning/Error: The clock source '<clock_source>' is the source of the reference of the generated clock '<gen_clock_source>' and require the specification of a correct injection node. Use 'set_clock_option <clock_source> -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and memories sourced by '<clock_source>' are in the fanout of the DFT inject node. (DFT_C10-x)
Warning/Error: The branch clock '<clock_branch_node>' is sourced by the clock '<clock>' without going through its "dft_inject_node '<injection_node>'." (DFT_C10-x)
Warning/Error: The specified -dft_inject_node '<injection_node>' for clock '<clock_source>' has no source. (DFT_C10-x)
Error/Warning: The clock '<clock_source>' is the source of the branch clock '<clock_branch_node>' and require the specification of a correct injection node. Use 'set_clock_option <clock_source> -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and all memories sourced by '<clock_source>' are in the fanout of the DFT inject node. 3) There are no references of generated clock in the fanout of the DFT inject node. 4) There are no branch clocks in the fanout of the DFT inject node. (DFT_C10-x)
Error/Warning: The dft injection node '<injection_node>' is defined for the clock source '<clock_source_1>' and for the clock source '<clock_source_2>'. Only differential clock pairs can share the same injection node. (DFT_C10-x)
Warning/Error: The specified -dft_inject_node '<injection_node>' for clock '<clock_source>' has a constant value of <0/1>. (DFT_C10-x)

Examples

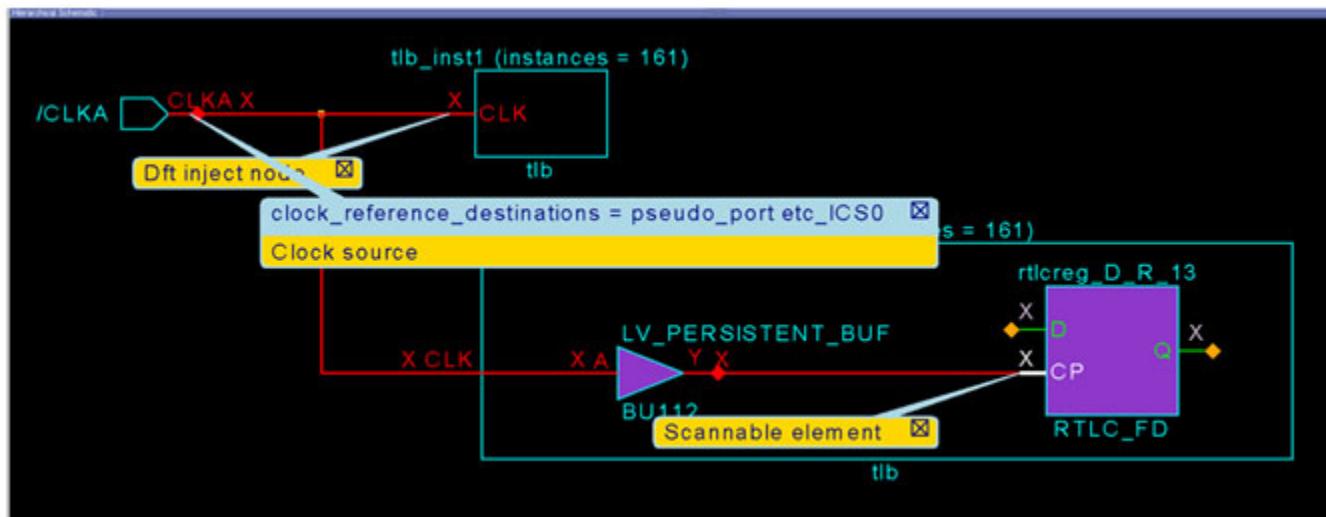
Example 1

```
// Error: The backward tracing from the specified -dft_inject_node
//        'buf_02/Y' did not reach the clock 'clkb'. The tracing stopped
//        at node 'rtlcreg_ff_clkb/Q'. (DFT_C10-6)
```



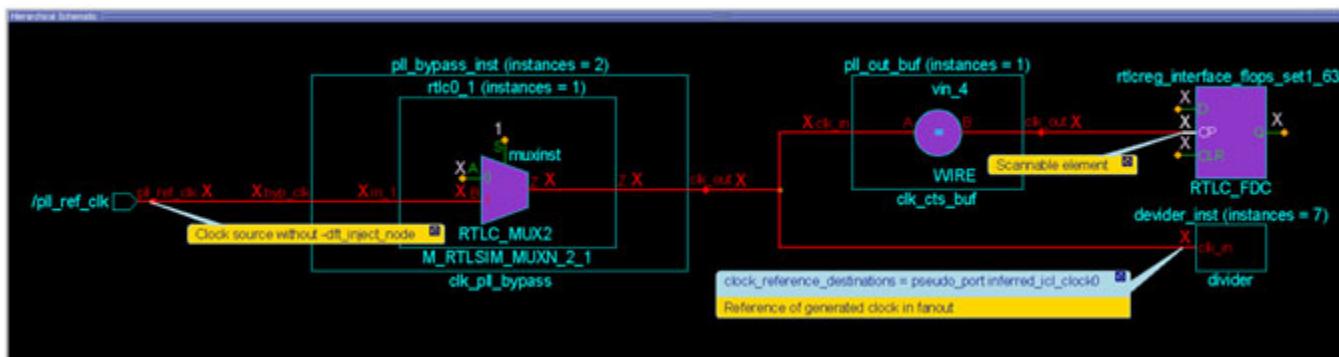
Example 2

```
// Error: The clock 'CLKA' fanouts to 156 scannable elements which are
// not the fanout of the specified -dft_inject_node 'tlb_inst1/CLK',
// which are:
//      'tlb_inst2/rtlcreg_D_R_13/CP'
//      'tlb_inst2/rtlcreg_D_R_12/CP'
//      'tlb_inst2/rtlcreg_D_R_11/CP'
//      'tlb_inst2/rtlcreg_D_R_10/CP'
//      'tlb_inst2/rtlcreg_D_R_9/CP'
//      'tlb_inst2/rtlcreg_D_R_8/CP'
//      'tlb_inst2/rtlcreg_D_R_7/CP'
//      'tlb_inst2/rtlcreg_D_R_6/CP'
//      'tlb_inst2/rtlcreg_D_R_5/CP'
//      'tlb_inst2/rtlcreg_D_R_4/CP'
//      ...
//      ... (DFT_C10-1)
```



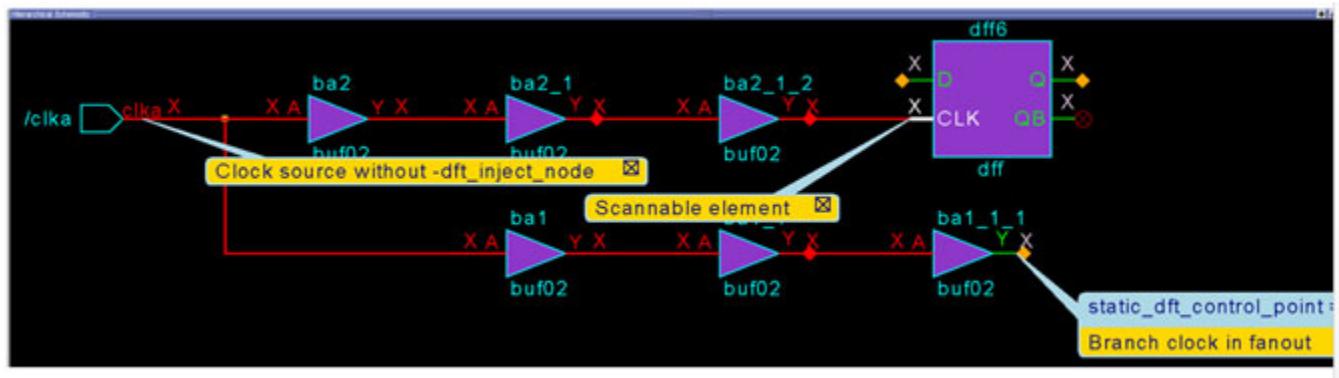
Example 3

```
// Error: The clock 'pll_ref_clk' is the source of the reference of the
// generated clock 'devider_inst/clk_out2' and requires the
// specification of a correct injection node.
// Use 'set_clock_option pll_ref_clk -dft_inject_node' to specify
// a node that matches the following criteria:
// 1) The DFT inject node is in the controlling fanout of
//    'pll_ref_clk'.
// 2) All scan elements and all memories sourced by
//    'pll_ref_clk' are in the fanout of the DFT inject node.
// 3) There are no references of generated clock in the fanout of
//    the DFT inject node.
// 4) There are no branch clocks in the fanout of the DFT inject
//    node. (DFT_C10-1)
```



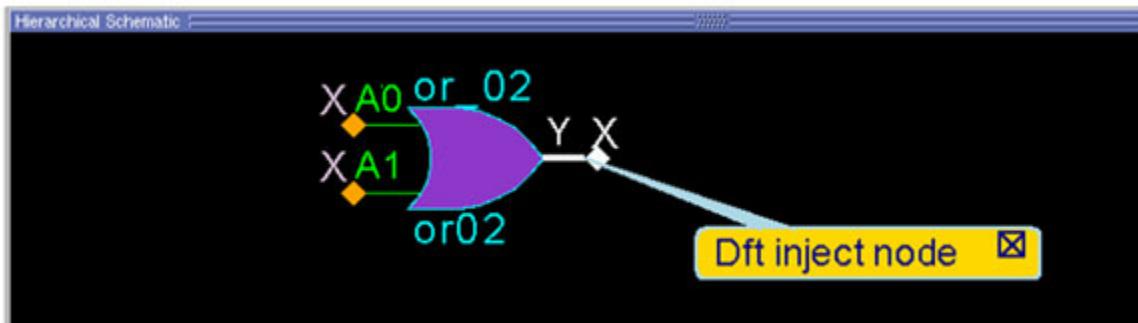
Example 4

```
// Error: The clock source 'clka' is the source of the branch clock
// 'ba1_1_1/Y' and require the specification of a correct
// injection node.
// Use 'set_clock_option clka -dft_inject_node' to specify a node
// that matches the following criteria:
// 1) The DFT inject node is in the controling fanout of 'clka'.
// 2) All scan elements and all memories sourced by 'clka' are in
//    the fanout of the DFT inject node.
// 3) There are no references of generated clock in the fanout of
//    the DFT inject node.
// 4) There are no branch clocks in the fanout of the DFT inject
//    node. (DFT_C10-1)
```



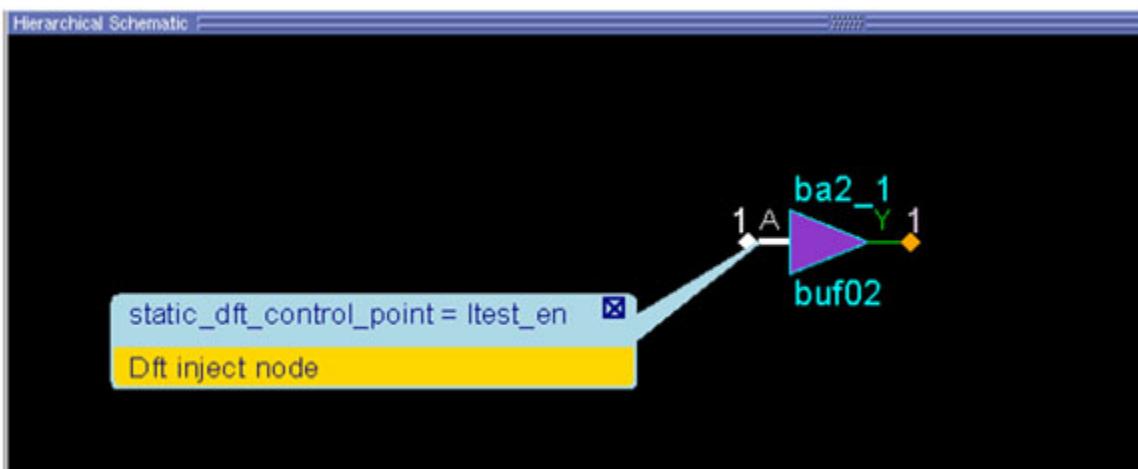
Example 5

```
// Error: The backward tracing from the specified -dft_inject_node
//          'or_02/Y' did not reach the clock 'clk1'. The tracing stopped
//          at node 'or2/Y'. (DFT_C10-2)
```



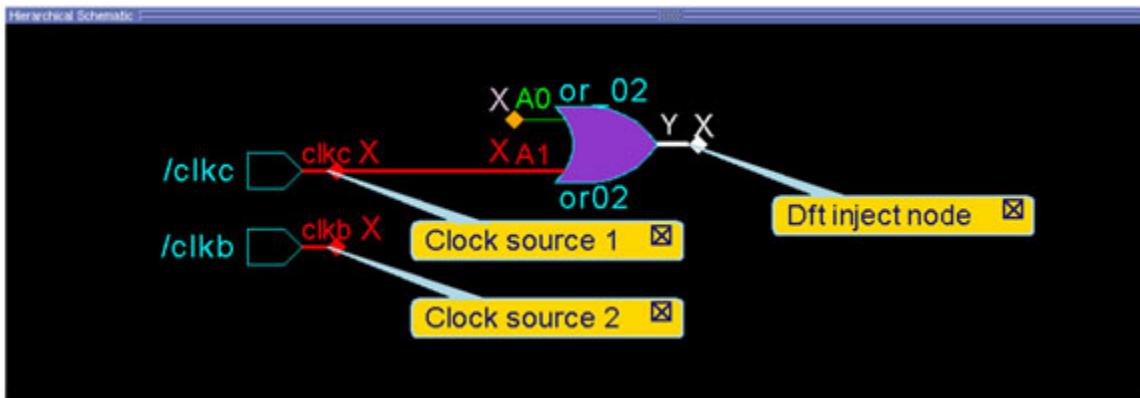
Example 6

```
// Error: The specified -dft_inject_node 'ba2_1/A' for clock 'clka' has a
//          constant value of 1. (DFT_C10-1)
```



Example 7

```
// Error: The dft injection node 'or_02/Y' is defined for the clock
// source 'clkc' and for the clock source 'clkb'.
// Only differential clock pairs can share the same injection
// node. (DFT_C10-1)
```



DFT_C11

Category: Clock

Context Supported: dft

Default Handling: Warning

report_drc_rules: Supported

Identifies and verifies the clock location and polarity for TCK mode injection. The default handling of this rule violation is warning, but it can be downgraded to note.

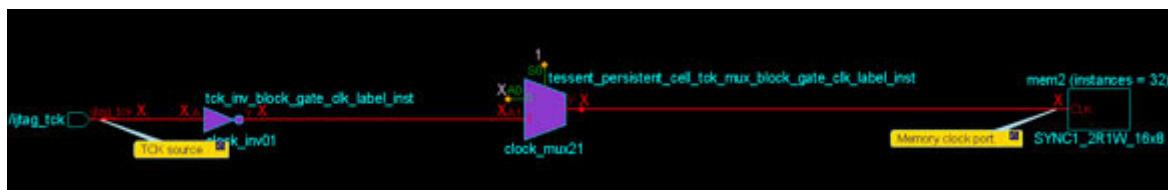
This DRC rule identifies clocks requiring the injection of TCK to run TCK mode patterns on all memories and child blocks. It also identifies existing TCK injection that is controlled by the tck_select DFT signal. The polarity of the existing TCK injection will be verified.

This rule check is executed only when the -tck_injection option of the `set_dft_specification_requirements` command is specified to “on” or “internal_sources_only”. This DRC rule is performed using the `dft_glb_ns_fe_tck` simulation context.

The warnings reported by this DRC rule are shown in [Table 7-11](#) and illustrated in the figure below.

Table 7-11. Warning Messages Generated by DFT_C11 Checks

The TCK clock that reaches the clock pin <clock_location> for TCK mode is inverted to the source. During TCK mode tests, memory BIST instruments require to be driven by a TCK that has a consistent polarity. The BAP, controllers and memory interfaces must receive an identical TCK clock. The polarity of TCK reaching the memory itself does not have to be consistent with the memory BIST instruments.



This warning reports existing TCK injection that could be incompatible with Tessent Shell's TCK mode. It is required that the TCK clock injected on the functional clock path, driving the memory BIST controllers and memory interfaces, be of the same polarity as the IJTAG network's TCK clock driving the memory BIST BAP. Since the polarity of the network's TCK clock is not yet known, this warning is issued.

DFT_C12

Category: Clock

Context Supported: dft

Default Handling: Error

report_dft_rules: Supported

Verifies that it is possible to edit the identified TCK injection points and verifies the direction for internal pin cases. The default handling of this rule violation is error, but it can be downgraded to note.

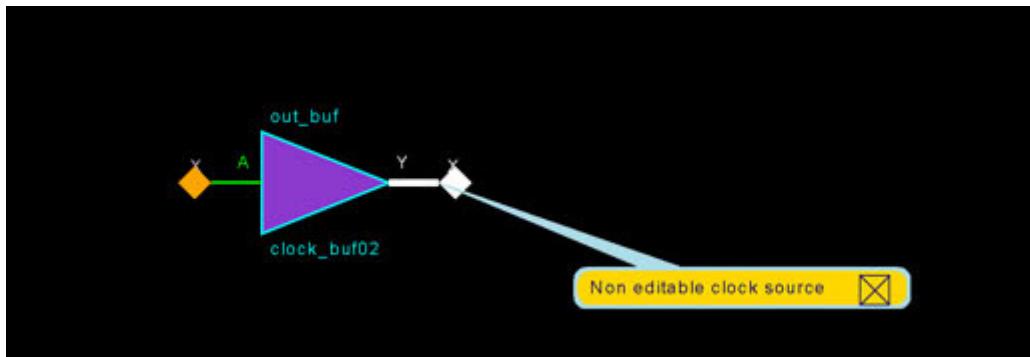
This rule check is executed only when the -tck_injection option of the `set_dft_specification_requirements` command is specified to “on” or “internal_sources_only”. This DRC rule is performed on the data collected during [DFT_C11](#).

Two types of errors are generated by this DRC rule:

The clock source <node_name> would require TCK injection but is not editable. Use the “-dft_inject_node” option of the `add_clocks` command to specify an editable location.

The clock source <clock_name> is pointing to input pin <node_name>. This is currently not supported for TCK injection. Define clocks on output pins or use the “-dft_inject_node” option of the `add_clocks` command to specify an output pin.

The first rule violation, illustrated in the figure below, occurs if a clock source requires the insertion of a clock mux and possibly an inverter for TCK injection, but that location is not editable.



The solution for this violation is to specify the first editable location following the clock source using the “-dft_inject_node” option of the [add_clocks](#) command. The other option is to downgrade the handling of this DRC, which will remove the violating locations reported from the [TCKInjectionPoints](#) wrapper during [create_dft_specification](#). The impact of this choice is that for TCK mode patterns, you will be required to inject TCK on the functional clock primary input port, bypassing any PLL or divider on the clock path.

The second error type reports a current limitation with TCK injection. TCK can only be injected on clocks defined on internal output pins. The solution is to define the clock on an output pin or to define an output pin using the “-dft_inject_node” option of the [add_clocks](#) command. If you choose to downgrade the handling of this DRC, the problematic <node_name> locations will be omitted from the [TCKInjectionPoints](#) wrapper during [create_dft_specification](#). The impact of this choice is that for TCK mode patterns, you will be required to inject TCK on the functional clock primary input port, bypassing any PLL or divider on the clock path.

DFT_C13

Category: Clock

Context Supported: dft

Default Handling: Error

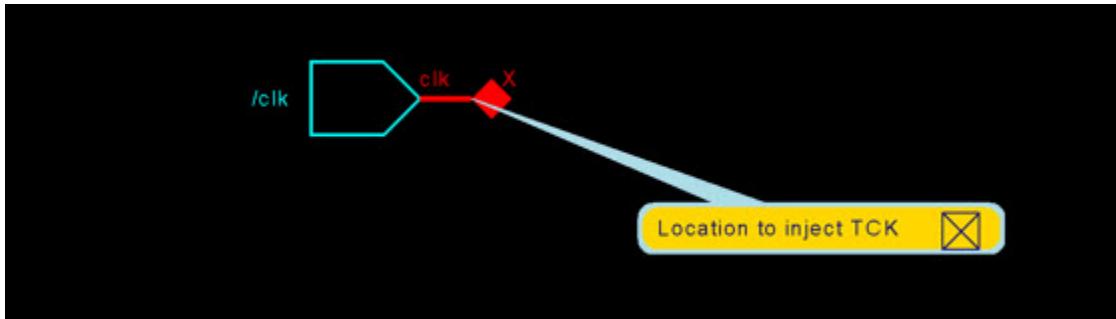
report_dft_rules: Supported

This rule verifies the existence or the scheduled addition of the required tck_select DFT signal.

This DRC rule is performed on the data collected during the [DFT_C12](#) rule check. DFT_C13 is executed only when the -tck_injection option of the [set_dft_specification_requirements](#) command is specified to “on” or “internal_sources_only”, and when clock locations have been identified for TCK injection.

The following error is generated if the tck_select DFT signal is not found, and illustrated in the figure below:

Clocks have been identified for TCK injection but the DFT signal tck_select does not exist. It is required to control the select of the inserted muxes. You must add the tck_select DFT signal to continue with the -tck_injection option of set_dft_specification_requirements.



The tck_select DFT signal is required to drive the select pin of the TCK injection muxes that will be inserted for memory BIST TCK mode. The solution for a DFT_C13 violation is to use the [add_dft_signals](#) command to add the tck_select signal. If it is chosen to downgrade the handling of this DRC to ignore, no TCK injection point will be identified and exported to the DftSpecification [TCKInjectionPoints](#) wrapper.

Extra Rules (E Rules)

Extra rules identify potential design requirement problems.

The following subsections describe the extra rules.

Tip

-  In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see “[How to Report Gate Data](#).”
-

E1.....	2775
E2.....	2776
E3.....	2776
E4.....	2777
E5.....	2780
E6.....	2782
E7.....	2783
E8.....	2783
E9.....	2784
E10.....	2785
E11.....	2790
E12.....	2791
E13.....	2791
E14.....	2792
E15.....	2795

E1

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

All scan cells must be LSSD scan cells that contain a master and a slave latch. They may also contain shadow latches. This rule is meant to enforce a strict LSSD architecture within a design. The application performs this check by inspecting the memory elements of all scan cells. The rule violation occurs when any memory element (that is not a latch or a scan cell) does not contain a SLAVE.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

MASTER N (G) is not an LSSD latch. (E1-1)

N is the instance name of the MASTER gate, and G is the gate ID number.

The summary message is:

N scan cells are not LSSD. (E1)

N is the number of occurrences of rules violation E1.

E2

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

There must be no data inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell. The application performs this check by inspecting the inversion data for all scan cells. The rule violation occurs when any adjacent scan cells (including SCI and SCO) have an inversion difference.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

Scan chain has inversion between N1 (G1) and N2 (G2). (E2-1)

N1 is the instance name of one MASTER (or SCI) gate, G1 is its gate ID number, N2 is the instance name of the other MASTER (or SCO) gate, and G2 is its gate ID number.

The summary message is:

There were N scan chain inversions. (E2)

N is the number of occurrences of rules violation E2.

E3

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

There must be no inversion between MASTER and SLAVE for any scan cell. The application performs this check by inspecting the inversion data for the memory elements of all scan cells. The rule violation occurs when the MASTER is inverted relative to its SLAVE.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

```
SLAVE N (G) is inverted relative to MASTER. (E3-1)
```

N is the instance name of the SLAVE, G is its gate ID number, and E3 is the rule ID number.

The summary message is:

```
There were N SLAVES inverted relative to MASTER. (E3)
```

N is the number of occurrences of rules violation E3.

E4

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Checks for conflicting values driving the same net during evaluation of the test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures. The tool performs this check using the simulated values of each time period of all test procedures except the capture procedure. A rule violation occurs if any bus gate is at an X state and two or more of its inputs are not at Z.

The default handling for this rule violation is warning.

Effect on Testability

Failure to satisfy this rule does not affect coverage, but will result in the risk of bus contention.

How to Debug E4 Violations

The occurrence message is:

```
Bus contention on N (G) occurred at time T of P procedure. (E4-1)
```

N is the instance/net name of the bus gate, G is the gate ID number, T is the time period, and P is the procedure name.

The summary message is:

There were N occurrences of bus contention in test procedures. (E4)

N is the number of times an E4 violation occurred.

You can access the simulated values at the gate where the error occurred by issuing the set_gate_report command with the Drc_pattern argument, then using the report_gates command to report the gate whose ID number is displayed in the error message. If this is the only DRC violation, or if you have changed the default handling to error, you can also view simulated values of the gate of interest by issuing set_gate_report with the Error_pattern argument prior to using report_gates. Gate reporting with the proper simulation data helps you identify the conflicting inputs; by tracing back from these inputs, you can identify how to correct the problem.

If you see an “A” appended to the “E4” in the occurrence message (E4-1-A for example), it indicates the occurrence is due to the tool aborting the E4 check before it was completed. As these are “possible” but unproven E4s, the tool may be able to screen out some of them if you raise the abort limit. Refer to “[Possible Resolutions](#)” for more information.

How to Debug with DFTVisualizer

To view the location of an E4 DRC violation using DFTVisualizer, use the following command steps:

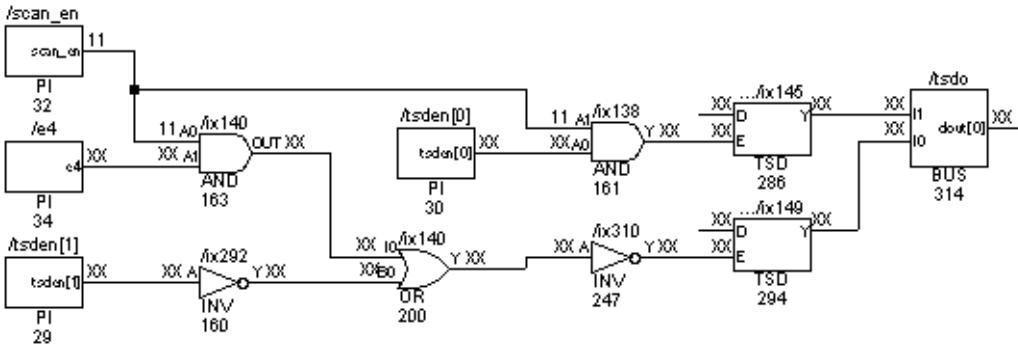
1. set_gate_level Primitive
2. open_visualizer

Choose the **DRC Violations** tab in the Design Browser window, then open the listed DRC’s by selecting the “+” symbols as needed to find the violation ID of the desired DRC violation. Double clicking the listed violation, or right clicking and selecting **Analyze DRC Violation** from the popup menu will analyze the violation. The tool will automatically change the gate reporting to “drc_pattern load_unload” and display the bus gate affected by that violation in the Flat Schematic window of DFTVisualizer.

Tip  Alternatively, you can issue the [analyze_drcViolation](#) command with the rule_id-occurrence# and -Display arguments at the tool’s command line. For example, to display the first occurrence of an E4 violation: analyze_drcViolation e4-1 -display

Once the offending bus gate is displayed, trace backward from the bus gate. [Figure 7-32](#) shows an example DFTVisualizer display of such a trace, where the starting point of the trace is the offending gate, /tsdo.

Figure 7-32. Example E4 Violation Trace in Flat Schematic Window of DFTVisualizer



How to Debug from the Tool Command Line

To view the location of the first occurrence of an E4 violation, use the following steps:

1. set_system_mode analysis
2. report_drc_rules E4-*occurrence#*
3. set_gate_level Primitive
4. set_gate_report Drc_pattern Load_unload
5. report_gates *gate_id*

The following transcript excerpts show an example of the use of this command sequence starting at step 2:

```
report_drc_rules e4-1
// Warning: Bus contention on /tsdo/dout [0] (314) occurred at time 0 of
// load_unload procedure. (E4-1)
set_gate_level primitive
set_gate_report drc_pattern load_unload
report_gates 314
// /tsdo (314) BUS
//     "I0"    I (XX)  294-/tsdo/ix149/Y
//     "I1"    I (XX)  286-/tsdo/ix145/Y
//     "OUT"   O (XX)  396-/dout [0]  330-
```

The Xs on the two inputs to the bus gate in this example are the source of the problem. A reasonable way to proceed is to choose one input and trace back through the gates driving it until you find the source of the Xs. For example, you might first look at where the “I0” input comes from:

```
report_gates 294
// /tsdo/ix149 (294)  TSD
//     E      I (XX)  247-/ix310/Y
//     "D"    I (XX)  221-
//     Y      O (XX)  314-
```

Notice the Xs on the enable (E) input. This input is of particular interest because an X here results in an X on the TSD's output regardless of the value on its D input. So tracing back from the enable would be a logical next step:

```
report_gates -endpoints -backward 247
// -----
// Begin backward trace for gate /ix310 (247).
// -----
// /tsden[1] (29) PI
//     tsden[1] O (XX) 160-/ix292/A
// /scan_en (32) PI
//     scan_en O (11) 163-/ix140/A0 161-/ix138/A1 162-/senmux/ix9/A1
// /e4 (34) PI
//     e4 O (XX) 163-/ix140/A1
// Number gates in trace = 3.
```

If you could get a 0 on the net connected to this TSD's enable, it would remove the Xs on its output.

Possible Resolutions

Cell constraints and ATPG constraints (dynamic or static) have no impact on the E4 rule. Pin constraints can have an impact because the tool may add extra cycles at the end of the test_setup procedure if the procedure does not preserve the constrained value. The constrained value can be overridden by another force value in the test procedure.

It is common during test_setup, when a design is being initialized, to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention and as a result, the tool may report many E4 violations for the test_setup procedure. If you know these momentary occurrences of bus contention are not a problem and do not wish to be reminded of them, you can disable the E4 checks for the test_setup procedure by issuing the command “set_drc_handling -Skip_procedure Test_setup.”

Increasing the abort limit with the set_abort_limit command can help screen out false E4 violations by enabling the tool to perform more extensive contention checking. This works for E4A violations only.

E5

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note in patterns -scan contexts only. Ignore otherwise.

report_drc_rules: Supported

When the application places constrained states on constrained pins and binary states on PIs and scan cells, X states must not propagate to an observable point. Failure to satisfy this rule will result in the risk of X states propagating to an observable point. This is a serious condition for

BIST circuits, may reduce the effective compression obtainable with EDT, and has no effect on uncompressed ATPG. For compressed designs, the number of E5 violations should be minimized in order to reduce the impact on compression.

Note

 Tessent TestKompress uses X-masking to handle the X states, but they typically increase the number of patterns generated, lowering the effective compression.

The application performs this check on gates that can create an X state with their inputs at binary values. It will not consider gates that do not have a path to an observable point, or that have all paths blocked by tied or constrained circuitry. The tool checks for the following conditions:

- A violation on a wired gate (WIRE) occurs if the tool can place different values on its inputs and the net resolution is set to wire.
- A violation on a BUS gate occurs if more than one of the BUS-connected tri-state drivers or switches turn on simultaneously, or all drivers turn off simultaneously and the Z state behaves as an X.
- A violation on a tri-state driver gate (TSD) or a switch gate (SW) occurs if it does not connect to a BUS gate. You can turn off the enable line, and the Z state behaves as an X.
- A violation on a TIE-X gate occurs if the gate is locally sensitizable up to the point where it has multiple fanouts (or observable points).
- A violation on a transparent latch (TLA) occurs if a single clock line is not set to its on-state, or the set and reset lines are not off.
- A violation on a ROM or RAM gate occurs if you can set a read line to off, the read_off value is X, and an output is sensitizable when the read line is off.
- A RAM/ROM violation also occurs if any memory element is uninitialized and an output is sensitizable to an observation point.
- PI pins can propagate Z to PO when Z handling is X (default). To prevent this, NO_Z constraints on PIs are considered by E5. Additionally, NO_Z constraints added in setup mode are not allowed to be deleted in a non-setup mode so the E5 check assumption is preserved.

The default handling for this rule violation in patterns -scan context is note; in other contexts, it is ignored. When a violation occurs, you can access the tied/constrained simulated values by setting the gate reporting to “constrain_value” and using the report_gates command for the gate ID number displayed in the DRC message. The tool needs to be in a non-setup system mode. By tracing back and forward from this gate, you can identify why the violation occurred.

The occurrence message is:

```
T gate N (G) may have an observable X-state. (E5-1)
```

T is the gate type, N is the instance/net name of the gate, and G is the gate ID number.

The summary message is:

N gates may have an observable X-state. (E5)

N is the number of occurrences of rules violation E5.

E6

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

When the tool places constrained states on constrained pins, the inputs of a gate must not have sensitizable connectivity to more than one memory element of a scan cell.

The tools consider a scan cell to have multiple memory elements if the cell contains, for example, a shadow element or copy element (refer to [Slave Element](#) and [Shadow Element](#) in the *Tessent Scan and ATPG User's Manual* for detailed information about these memory elements and their use in scan cells).

The application performs this check by tracing the forward cones of influence of all scan cell memory elements through unconstrained and untied circuitry. The rule violation occurs when any gate is in the cone of influence of more than one memory element of a single scan cell.

The default handling for this rule violation is ignore. Failure to satisfy this rule may result in some loss of test coverage, but most faults should be detectable using a skewed load test procedure.

The occurrence message is:

Multiple memory elements of scan cell P (C) are connected to N (G). (E6-1)

P is the position number of the scan cell, C is the chain name, N is the instance name of the gate, and G is its gate ID number.

The summary message is:

There were N scan cells with multiple memory element connectivity to a gate. (E6)

N is the number of occurrences of rules violation E6.

E7

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

External bidirectional drivers must be at the high impedance (Z) state during the application of the test procedures. You can use this rule to ensure that no bus contention can occur at bidirectional pins independent of the force values on the bidirectional pins. The application performs this check using the simulated values of each time period of all test procedures (except test_setup). The rule violation occurs if any bidirectional tri-state driver is not at a Z state. Using the -Mode option with the set_drc_handling command and the report_drc_rules command, you can check the value being forced on the bidirectional pins.

The default handling for this rule violation is ignore. If rule E4 (which ensures bus-mutual exclusivity) passes, a violation of rule E7 has no effect. Failure to satisfy this rule (normally) has no effect if there is no violation of rule E4, which ensures no bus contention actually occurs.

You can access simulated gate values using the report_gates command with the gate data set to DRC_pattern. If this is the only DRC violation or if you have changed the default handling to error, you can also view simulated values using report_gates with gate reporting set to error_pattern. Gate reporting with the proper simulation data helps you identify the bidirectional pin that failed, and by tracing connectivity from this point, you can identify how to correct the problem.

Note

 You should set the gate data prior to any violation analysis, to ensure the gate data reported is for the correct violation.

The occurrence message is:

```
BIDI pin P not set to input mode at time T of P procedure. (E7-1)
```

P is the pin name of the bidirectional pin, T is the time period, and P is the procedure name.

The summary message is:

```
There were N occurrences of BIDIs not set to input mode during scanning.  
(E7)
```

N is the number of occurrences of rules violation E7.

E8

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

All scan cell MASTER elements of a scan chain must use a single shift clock. If the scan cells contain slaves, all slaves of all scan cells of a scan chain must also use a single shift clock. You can use this rule to ensure that the tester does not cause clock skew problems during the loading and unloading of the scan chains. The application performs this check by inspecting the memory elements for all scan cells of a chain. The rule violation occurs when a chain uses multiple clocks to shift master or slave data.

If multiple shift clocks pulse in the shift procedure and they are blocked from reaching the scan chain by the effects of pin constraints, you must specify the set_drc_handling command with the Atpg_analysis option to avoid an error. This checking considers only pin constraints that have not been overridden by force statements during the shift and load_unload procedures.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no ATPG effect. The occurrence message is:

Multiple clocks were used to shift T of scan chain C. (E8-1)

T is the type of scan cell memory element (MASTERs or SLAVEs), and C is the chain name.

The summary message is:

There were N occurrences of multiply clocked scan chains. (E8)

N is the number of occurrences of rules violation E8.

Note

 If you set the DRC E8 handling to anything other than Ignore, the report_scan_chains command additionally displays the clock of each scan cell. This enhanced report is only available in non-Setup mode.

E9

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

The drivers of wire gates must not be capable of driving opposing binary values. The application performs this check by attempting to satisfy the placement of opposing binary values on all combinations of the two drivers of a wire gate. The rule violation occurs at a wire gate if it is possible to satisfy those conditions for at least one combination of drivers. When a

violation occurs, the tool identifies the failing wire gate and the drivers capable of being placed at opposing values.

This rule ensures that there is no possible contention (for the good machine) on wire gates. The tool will not perform this rule check on wire gates whose behavior you changed to AND or OR using the set_net_resolution command. Also, the tool does not consider pin constraints and equivalences with this check.

The default handling for a violation of this rule is set to ignore. A violation of this rule indicates the possibility that patterns exist that have contention on wire gates. The occurrence message is:

```
WIRE gate N (G) has possible contention on drivers G1 and G2. (E9-1)
```

N is the gate name of the wire gate, G is its gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N WIRE gates which may have possible contention. (E9)
```

N is the number of occurrences of rules violation E9.

E10

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Checks for bus contention mutual-exclusivity during measure_po events in test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the measurement of primary output (PO) values (measure_po events in test procedures). This rule differs from the E4 rule in that it is not checked during test procedure events other than measure_po events. The tool performs this check by analyzing each dominant strong bus to determine if it can cause contention. The analysis places each bus in one of the following categories:

- Pass - Test generation analysis determines a contention condition cannot occur.
- Fail - Test generation analysis identifies a possible contention condition.
- Abort - Test generation analysis terminated while attempting to determine if a contention condition could occur.
- Bidi - Test generation determines that the bidirectional pin (which can only have a single tri-state driver) can potentially create a contention condition.

Buses in either the fail or abort categories violate this rule. Buses in the bidi category require checking during post-DRC ATPG processes because they do not exhibit natural mutual-exclusivity behavior and can potentially cause contention. You get the tool to perform this additional analysis by using the Atpg_analysis option with the set_drc_handling command. Buses in the pass category require no further checking.

Refer to “[Bus Mutual Exclusivity Analysis](#)” in the *Tessent Scan and ATPG User’s Manual* for background information about the tool’s bus mutual exclusivity checking. For more information about the measure_po statement, refer to the “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

Effect on Testability

Failure to satisfy this rule will result in the risk of bus contention during measurement of PO values (measure_po events in test procedures).

How to Debug E10 Violations

The occurrence message is:

```
Bus gate N (G) has possible contention on drivers G1 and G2. (E10-1)
```

N is the gate name of the bus gate, G is the gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N bus gates which may have possible contention (E10)
```

N indicates the number of bus gates failing the E10 rule.

If you see an “A” appended to the “E10” in the occurrence message (E10-1-A for example), it indicates the occurrence is due to the tool aborting the E10 check before it was completed. As these are “possible” but unproven E10s, the tool may be able to screen out some of them if you raise the abort limit. Refer to “[Possible Resolutions](#).”

How to Debug with DFTVisualizer

To view the location of an E10 DRC violation using DFTVisualizer, use the following command steps:

1. set_gate_level Primitive
2. open_visualizer

Choose the **DRC Violations** tab in the Design Browser window, then open the listed DRC’s by selecting the “+” symbols as needed to find the violation ID of the desired E10 DRC violation. Double clicking the listed violation, or right clicking and selecting **Analyze DRC Violation**

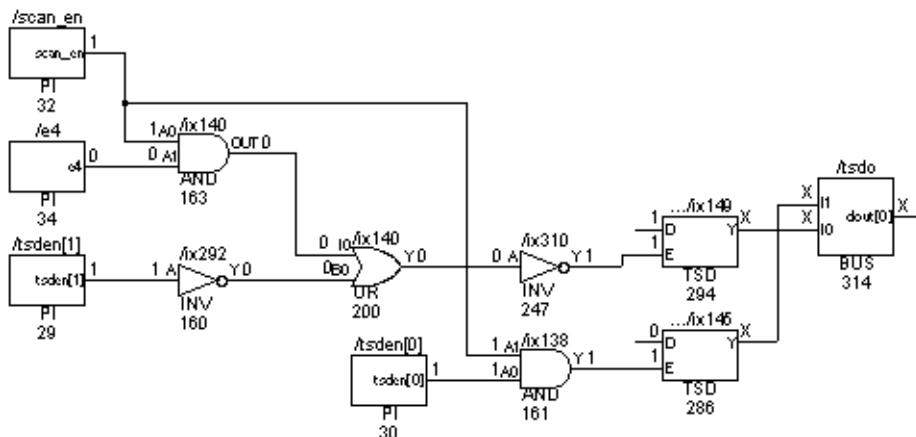
from the popup menu will analyze the violation. The tool will automatically change the gate reporting to “parallel_pattern 0” and display the bus gate affected by that violation in the Flat Schematic window of DFTVisualizer. Think of “parallel_pattern 0” as an example of one possible pattern (created for DRC purposes only) that, if included in a test pattern set, would cause contention.

Tip

i Alternatively, you can issue the `analyze_drcViolation` command with the `rule_id-occurrence#` argument at the tool’s command line. For example, to display the first occurrence of an E10 violation: `analyze_drcViolation e10-1`

Once the offending bus gate is displayed, trace backward from the bus gate. The following figure shows an example DFTVisualizer display of such a trace, where the starting point of the trace is the offending gate, `/tsdo`. Notice that TSDs `/ix145` and `/ix149` have conflicting values and both are enabled.

Figure 7-33. Example E10 Violation Trace in Flat Schematic Window of DFTVisualizer



You could use DFTVisualizer to trace back on the enable line of each TSD to the primary inputs and see that both `tsden[1]` and `/scan_en` are 1, causing the conflict.

How to Debug from the Tool Command Line

To view the location of the an occurrence of an E10 DRC violation from the tool command line, you can use the following command steps:

1. `set_drc_handling E10 Warning`
2. `set_system_mode analysis`
3. `report_drc_rules E10-occurrence#`
4. `analyze_bus gate_id`

5. set_gate_report Parallel_pattern *pattern_number*
6. report_gates *gate_id*

The following transcript excerpts show an example of the use of this command sequence starting at step 3:

```
report_drc_rules e10-1
// Warning: BUS gate /tsdo/dout[0] (314) has possible contention on
//           drivers 294 and 286. (E10-1)
analyze_bus 314
// ATPG bus checking performed on /tsdo/dout[0] (314) for 2 bussed TSDs.
// Failure occurred while checking TSDs 294 and 286 (data in
// parallel_pattern 0).
set_gate_report parallel_pattern 0
report_gates 294 286
// /tsdo/ix149 (294) TSD
//   E     I (1) 247-/ix310/Y
//   "D"   I (1) 221-
//   Y     O (X) 314-
// /tsdo/ix145 (286) TSD
//   E     I (1) 161-/ix138/Y
//   "D"   I (0) 213-
//   Y     O (X) 314-
```

The conflicting values on the input (“D”) of each TSD in this example while both are enabled (1 on the “E” input) is the source of the problem. Assuming the two TSDs should not be enabled simultaneously, a reasonable way to proceed is to choose one enable input and trace back through the gates driving it until you find the source of the 1. For example, you might first look at where the /tsdo/ix149 (294) input comes from:

```
report_gates -endpoints -backward 294
// -----
// Begin backward trace for gate /tsdo/ix149 (294).
// -----
// /tsden[1] (29) PI
//   tsden[1] O (1) 160-/ix292/A
// /scan_en (32) PI
//   scan_en O (1) 163-/ix140/A0 161-/ix138/A1 162-/senmux/ix9/A1
// /e4 (34) PI
//   e4 O (0) 163-/ix140/A1
// /dataot/reg_q_0_ (373) DFF
//   "S" I (X) 43-
//   R I (X) 157-
//   CLK I (X) 313-/dataot/ix104/Y
//   "DO" I (X) 255-
//   "OUT" O (1 [X]) 127- 128-
//   MASTER cell_id=8 chain=chain3 group=grp1 invert_data=TFFT
// Number gates in trace = 4.
```

This shows that /tsden[1] is a 1. A similar backward trace from instance 286 to /tsden[0] shows that it too is a 1. If you could get a 0 on one of the /tsden inputs, it would remove the contention.

Possible Resolutions

Be aware you may not need to care about E10 violations in certain cases. If you are not concerned about contention after the capture clock is pulsed and you are satisfied with test coverage, you do not need to do anything. On the other hand, if the tool is rejecting many patterns due to contention during simulation and as a result, run time is excessive and coverage low, you may want to do something. By default, the tool will check for contention prior to the capture clock pulse in the capture cycle. If you are concerned about contention after the capture clock is pulsed, you need to direct the tool to check for this type of contention. The key point to remember is you only need to do something if you are not happy with what the tool does by default or the run time or coverage is unacceptable, in which case the following commands are helpful:

Note

 These command suggestions will not make E10 rule violations go away, but will help you generate contention-free patterns when E10 violations are present.

- [add_input_constraints](#) — Adds a pin constraint to a primary input pin. This command is useful if you determine from your troubleshooting and knowledge of the design that forcing a particular primary input pin to a specific value would resolve a contention issue. Typically, the need for such a constraint would have been anticipated during the design phase and perhaps simply overlooked when setting up for ATPG.
- [add_atpg_functions](#) and [add_atpg_constraints](#) — These commands allow you to create user-defined ATPG function constraints. This can be useful if you know where in the design constraints can be applied to remove contention. Constraints you apply will save time the tool would have to spend learning those locations through its own analysis.

Following is an example of the use of these commands to specify constraints:

```
add_atpg_functions func1 select1 tsden[0] ~tsden[1]
add_atpg_constraints 1 func1
```

- [set_drc_handling E10 Atpg_analysis](#) — Directs the tool to perform additional analysis toward the end of DRC to determine if buses that the initial E10 DRC placed in the bidi category can cause contention. Increases the tool's DRC effort.
- [set_drc_handling E10 -Mode Sequential](#) — This command considers the inputs to a single level of sequential cells behaving as “staging” latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enable of the tri-state devices except through the sequential cells. Using this option will increase run time.

- **set_abort_limit** — Changes the abort limit. Increasing the abort limit can help screen out false E10 violations by enabling the tool to perform more extensive contention checking. This works for E10-*occurrence#*-A violations only.

E11

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

This rule checks for the ability of a bus gate to attain a Z state. This check analyzes each dominant strong bus to determine if conditions can place a Z value on the bus gate.

As a result, the analysis places each bus in one of the following categories:

- **Pass** - Test generation analysis determines that no condition could place a Z value on the bus gate.
- **Fail** - Test generation analysis determines that conditions could place a Z value on the bus gate.
- **Abort** - Test generation analysis terminated while attempting to determine if conditions could place a Z value on the bus gate.
- **Bidi** - The bus is a bidirectional bus.

Buses in both the fail and abort categories violate this rule.

The default settings for this rule are ignore, noverbose, and atpg_analysis. You can change the handling with the set_drc_handling command. For more information on ATPG analysis, refer to “[How to Turn on ATPG Analysis](#).”

The occurrence message is:

```
BUS gate N (G) is capable of attaining a Z state. (E11-1-A)
```

N is the gate name of the bus gate, and G is its gate ID number. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N BUS gates capable of attaining a Z state. (E11)
```

N indicates the number of buses failing the E11 rule.

E12

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

This rule determines if the test procedures violate any ATPG constraints. The default handling for this rule is ignore. You can change the handling with the set_drc_handling command.

For more information on ATPG analysis, refer to “[How to Turn on ATPG Analysis](#).”

The occurrence message is:

```
ATPG constraint violation on N (G) occurred at time T of P procedure.  
(E12-1-A)
```

N is the gate name, G is its gate ID number, T is the simulated time period, and P is the procedure name. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N occurrences of ATPG constraint violations in test procedures.  
(E12)
```

N indicates the number of E12 rule violations.

E13

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

This rule determines if it is possible to satisfy both ATPG constraints and bus contention prevention (for buses that fail rule E10). The default settings for this rule are ignore and noverbose. You can change the handling with the set_drc_handling command.

The occurrence message is:

```
Contention prevention/ATPG constraints satisfiability check failed. (E13-  
1)
```

This rule does not issue a summary message.

E14

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis,
patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

The expected simulation values listed in the test_setup procedure must match the actual simulated values.

E14 ensures the test_setup procedure is correct.

This DRC simulates the test_setup procedure when setup mode is exited and compares the simulated values with the expected values listed in the test_setup procedure.

Exception

Not applicable.

Messaging

If the test_setup procedure file fails E14, the following message displays:

```
// Error: Mismatch on <signal_name> at time <n>, cycle <cycle>
//          of procedure <proc>. Expected <val>, simulated <val>. (E14-#)
```

When the E14 violation originates from an iRead, it reports the original iRead target, the iCall that called the iProc containing the iRead, the chain flip-flop name if appropriate core data exists, and the iReadVar if applicable. The following message displays:

```
// Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//          procedure <proc>. Expected <val>, simulated <val>.
//          This compare event originates from
//          an iRead command targeting <iRead_target> in iCall <iCall_name>.
//          Use the command 'report_procedures -expand_iprocs' to help debug
//          failures within iProcs. (E14-1)
```

If an ICL ScanRegister has a capture source with known values (that is, if the bits of the CaptureSource property consist of constant values, references to primary inputs, or references to the parallel outputs of other ScanRegister bits with known content) and there is no explicit iRead targeting the Scan Register, then Tesson Shell creates an implied iRead for each of those ScanRegister bits with known capture source. The implied iRead commands are created only

for the scan loads in which this ScanRegister is active. If the E14 violation originates from such an implied iRead, one of the following messages displays:

```
// Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
// procedure <proc>. Expected <val>, simulated <val>.
// This compare event originates from an implied iRead command
// targeting <register> (<register> captures the constant value <val>)
// in iCall <iCall_name>.
// Use the command 'report_procedures -expand_iprocs' to help debug
// failures within iProcs. (E14-1)

// Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
// procedure <proc>. Expected <val>, simulated <val>.
// This compare event originates from an implied iRead command
// targeting <register> (<register> captures the value from the
// scan register <other_register>) in iCall <iCall_name>.
// Use the command 'report_procedures -expand_iprocs' to help debug
// failures within iProcs. (E14-1)

// Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
// procedure <proc>. Expected <val>, simulated <val>.
// This compare event originates from an implied iRead command
// targeting <register> (<register> captures the value from the
// primary input <primary_input>) in iCall <iCall_name>.
// Use the command 'report_procedures -expand_iprocs' to help debug
// failures within iProcs. (E14-1)
```

Tessent Shell also verifies the iWrite commands in the test_setup and test_end procedures, if possible. During the processing of an iWrite command that targets an ICL pin, Tessent Shell tries to map the ICL pin to a design pin using the “tessent_design_instance” attributes of the relevant ICL instances. The iWrite verification of a particular ICL pin is only done if all “tessent_design_instance” attributes on the ICL instance path from the top level to the targeted ICL pin are set. As the tessent_design_instance attribute usually contains the rtl name of the instance, Tessent Shell also uses the rtl-to-gate mapping mechanism (see the [add_rtl_to_gates_mapping](#) command) to find the design pin in the gate level netlist. If this mapping fails, the following warning appears:

```
// Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
// targeting the ICL object <icl_object>, which has been resolved
// to the driving ICL pin <icl_pin>. The effect of this iWrite
// command cannot be verified during the E14 check. The rtl path
// <rtl_path> of the associated design pin, which has been composed
// from all 'tessent_design_instance' attributes on the ICL instance
// path to the ICL pin, cannot be mapped to an existing design pin
// in the netlist.
// Either the netlist is incomplete or the mapping between rtl
// description and gatelevel netlist failed. If this problem
// originates from a mapping failure, use the command
// 'add_rtl_to_gates_mapping -instance_name_map_list' to
// specify the appropriate instance mapping.
```

If the design pin can be found and it is tied off to a value that conflicts with the iWrite command, the following warning appears:

```
// Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
// targeting the ICL object <icl_object>, which has been resolved
// to the driving ICL pin <icl_pin>. The effect of this iWrite
// command cannot be verified during the E14 check, because the
// associated design pin <design_pin> is tied to <val>, while the
// iWrite command tries to apply the value <other_val> to this pin.
```

If the design pin is an internal input with a constraint that conflicts with the iWrite command, the following warning appears:

```
// Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
// targeting the ICL object <icl_object>, which has been resolved
// to the driving ICL pin <icl_pin>. The effect of this iWrite
// command cannot be verified during the E14 check, because the
// associated design pin <design_pin> is constrained to <val>, while
// the iWrite command tries to apply the value <other_val> to this pin.
```

When the E14 violation originates from the verification of an iWrite command, it reports the original iWrite target and the iCall that called the iProc containing the iWrite. The following message displays:

```
// Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
// procedure <proc>. Expected <val>, simulated <val>.
// This compare event originates from the verification of an
// iWrite to <iWrite_target> during test
// setup in iCall <iCall_name>.
// Use the command 'report_procedures -expand_iprocs' to help debug
// failures within iProcs. (E14-1)
```

The summary message for E14:

```
E14: #fails=1 handling=warning (Miscompare on test_setup/test_end values)
```

Troubleshooting

1. Use DFTVisualizer to compare the DRC simulated values from the test_setup procedure with the simulated values for associated nodes in the design.
2. Correct the expect statements in the test_setup procedure to match the DRC simulation values.
3. For iMerges, both the [write_procfile](#) command and the [report_procedures](#) commands have an optional switch (-EXPand_iprocs) to allow for the expansion of the iCalls within a given procedure. If an iCall is used within an iMerge, then that entire iMerge is expanded.

E15

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis,
patterns -scan_retargeting

Default Handling: Warning

report_drc_rules: Supported

This DRC checks if the PDL-based test setup is stable until the end of the test_setup procedure.

If the test_setup procedure contains iCall statements or iCalls added by means of the `set_test_setup_icall` command, and if there are iWrite commands in the iProcs referenced by the iCalls, then Tesson Shell creates the required patterns to ensure that the iWrite values are present at the specified locations in the design. The tool identifies the values which can be expected to persist until the end of the test_setup procedure, only considering the ICL and PDL input. Finally, E15 checks if these relevant iWrite values match the results of the test_setup simulation in the stable_after_setup simulation context. As the DRC compares the results of the IJTAG-based value tracking with the results of the design-based test_setup simulation, a failure indicates either that the IJTAG-based test setup has been corrupted by other (non-IJTAG related) actions in the test_setup procedure, or that there are inconsistencies between the design and the ICL description of the IJTAG network.

Like the iWrite verification of the E14 DRC, the E15 DRC also requires a successful mapping between ICL pins and design pins. The same conditions (presence of the "tesson_design_instance" attributes and so on) apply. See the description of E14 for details.

Messaging

If the test_setup procedure file fails E15, the following message displays:

```
// Error: Mismatch on pin '<pin>' at the end of the test_setup procedure.  
// Expected '<val>', simulated '<val>'.  
// During test_setup, the value '<val>' has been applied to ICL element <icl elem>  
// by means of an iWrite command in 'iCall <icall>'.  
// This value is still expected to be present at this pin at the end  
// of test_setup as it has not been overwritten by subsequent iWrite commands  
// or other PDL retargeting activities. (E15-#)
```

EDT Finder Rules (F Rules)

F rules verify the EDT logic connectivity and operation prior to scan chain tracing.

F3	2796
F4	2797
F5	2797
F6	2798
F7	2800
F8	2800
F9	2806
F10	2807
F11	2809
F12	2809
F13	2809
F14	2810
F15	2811
F16	2811
F17	2812
F18	2813
F19	2813
F20	2814
F21	2816
F22	2817

F3

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

The design must contain at least one state element able to change states during shift. F3 verifies that at least one state element in the design is clocked and not constrained to a fixed value.

You cannot change the handling with the set_drc_handling command.

The error message is:

At least one state element should be able to change its state during shift. None were found. (F3)

F4

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT requires a design containing at least one decompressor. F4 verifies there is a decompressor in the design.

You cannot change the handling with the set_drc_handling command.

The error message is:

No decompressors found. (F4)

F5

Category: EDT

Contexts Supported: patterns -scan

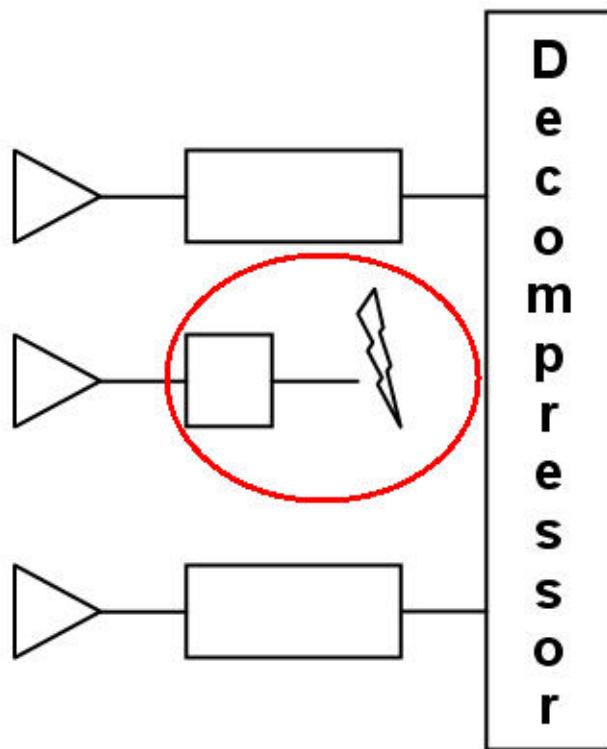
Default Handling: Error

report_drc_rules: Supported

All defined channel inputs must drive a decompressor. F5 verifies that all defined input channels are driving an identified decompressor by tracing the path from the channel input to the decompressor.

[Figure 7-34](#) shows an error condition.

Figure 7-34. Blocked Decompressor Input



The error message is:

Channel input P (N) [of EDT block E] is not driving a decompressor.
Either no structural connection exists, or the path is blocked.
Trace stopped at state element(s) S (G). (F5-1)

Where P is the channel index and N the pin name, E is the name of an EDT block, S is an instance name and G is the corresponding gate ID number. Multiple state elements may be listed in case of fan-outs.

Troubleshooting

Check the procedure file for incorrect or missing shift procedure settings.

F6

Category: EDT

Contexts Supported: patterns -scan

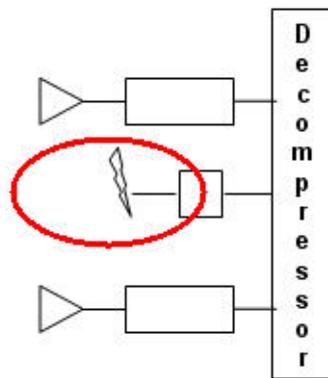
Default Handling: Warning

report_drc_rules: Supported

Channel input pins that control the decompressors must be defined. F6 traces each decompressor input pipeline to verify that decompressors are controlled by channel input pins, and verifies that the state elements in the channel input pipelines are properly clocked during shift.

Figure 7-35 shows an error condition.

Figure 7-35. Blocked Channel Input to Decompressor



This check is performed even when F5 violations exist.

If the decompressor is driven by one or more defined channel inputs, each blocked input pipeline is reported as follows:

The input pipeline of decompressor D [of EDT block E] is broken between decompressor state element L (G1) and the channel input after tracing N state elements. The last state element reached is: M (G2). (F6-1)

If all input pipelines of a decompressor are blocked the following message is reported once per decompressor.

All input pipelines of decompressor D are blocked. One of the input pipelines is broken between decompressor state element L (G1) and the channel input after tracing N state elements. The last state element reached is: M (G2). (F6-1)

If the clock input of a channel input pipeline is not set active during shift, the following message is reported:

The input pipeline of decompressor D [of EDT block E] is broken between decompressor state element L (G1) and the channel input after tracing N state elements. The clock inputs of state element M are never set active during the shift procedure (F6-1)

Where D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G1 is the corresponding gate ID number, N is the number of found pipeline stages, M is the instance name of the last traced state element and G2 is its gate ID number.

F7

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Decompressors must be controlled by defined channel inputs. F7 traces all inputs to the decompressor to verify that they are connected to channel input pins.

If the decompressor is driven by at least one defined channel input, each rule violation is reported as follows:

The pin P is not a channel input but drives decompressor state element L (G) of decompressor D [of EDT block E]. (F7-1)

If the decompressor is only driven by pins not defined as channel inputs, the following error message is reported once per decompressor:

None of the primary input pins driving decompressor D is defined as a channel input. One of the traced pins is P. (F7-1)

Where P is a pin name, D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G is the corresponding gate ID number.

Note

 In the EDT mapping flow, the reported pins are channel inputs mapped to the top level of the design.

F8

Category: EDT

Contexts Supported: patterns -scan

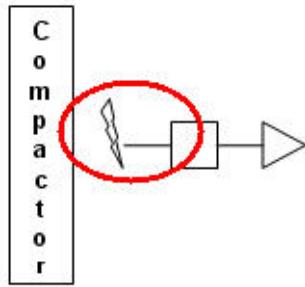
Default Handling: Error

report_drc_rules: Supported

Verifies that all channel outputs are driven by a compactor. F8 traces all channel outputs to verify that they are driven by a compactor, and verifies that the state elements in the channel output pipelines are properly clocked during shift.

Figure 7-36 shows an error condition.

Figure 7-36. Output Channel and Compactor Connection Blocked



The error message is:

Channel output P (N) [of EDT block E] is not driven by a compactor. Either no structural connection exists, or the path is blocked. The last state element reached while tracing back is: M (G). The following gate drives the channel output and blocks the path: M (G) (F8-1)

If the clock input of a channel output pipeline is not set active during shift, the following message is reported:

Channel output P (N) [of EDT block E] is not driven by a compactor. Either no structural connection exists, or the path is blocked. The clock inputs of state element M (G) are never set active during the shift procedure (F8-1)

If a primary input was reached the message is:

The primary input pin PI was reached while tracing back from channel output P (N) [of EDT block E]. (F8-1)

If an unclocked element is driving the channel output, the message is:

N state elements drive the channel output but could not be identified as mask hold registers (because they are not driven by mask shift registers) nor as part of scan chains (because the clock inputs are never set active during the shift procedure). One of those is: M1 (G1).

...
N is the number of unclocked state element driving the channel output,
M1 is the instance name of one of those, G1 is its gate ID number.

If the tracing problem is caused by a defined value on a channel output, the message is:

The channel output P (N) [of EDT block E] cannot be traced back because it is constrained to V. An unconstrained simulation value of X is expected. (F8-1)

If the tracing problem is caused by wrong values in mask hold registers, the message is:

```
Wrong values detected in mask hold registers after disabling EDT X masking logic and simulating load_unload until first clock event in apply shift. One of the failing state elements is: M (G). Simulated value: V1. Expected value: V2. (F8-1)
```

Where P is the channel index and N is the pin name, E is the name of the EDT block, M is the instance name of the last traced state element, G is its gate ID number, PI is the name of the reached primary input pin, and V, V1, V2 are gate values.

How to Debug Using DFTVisualizer

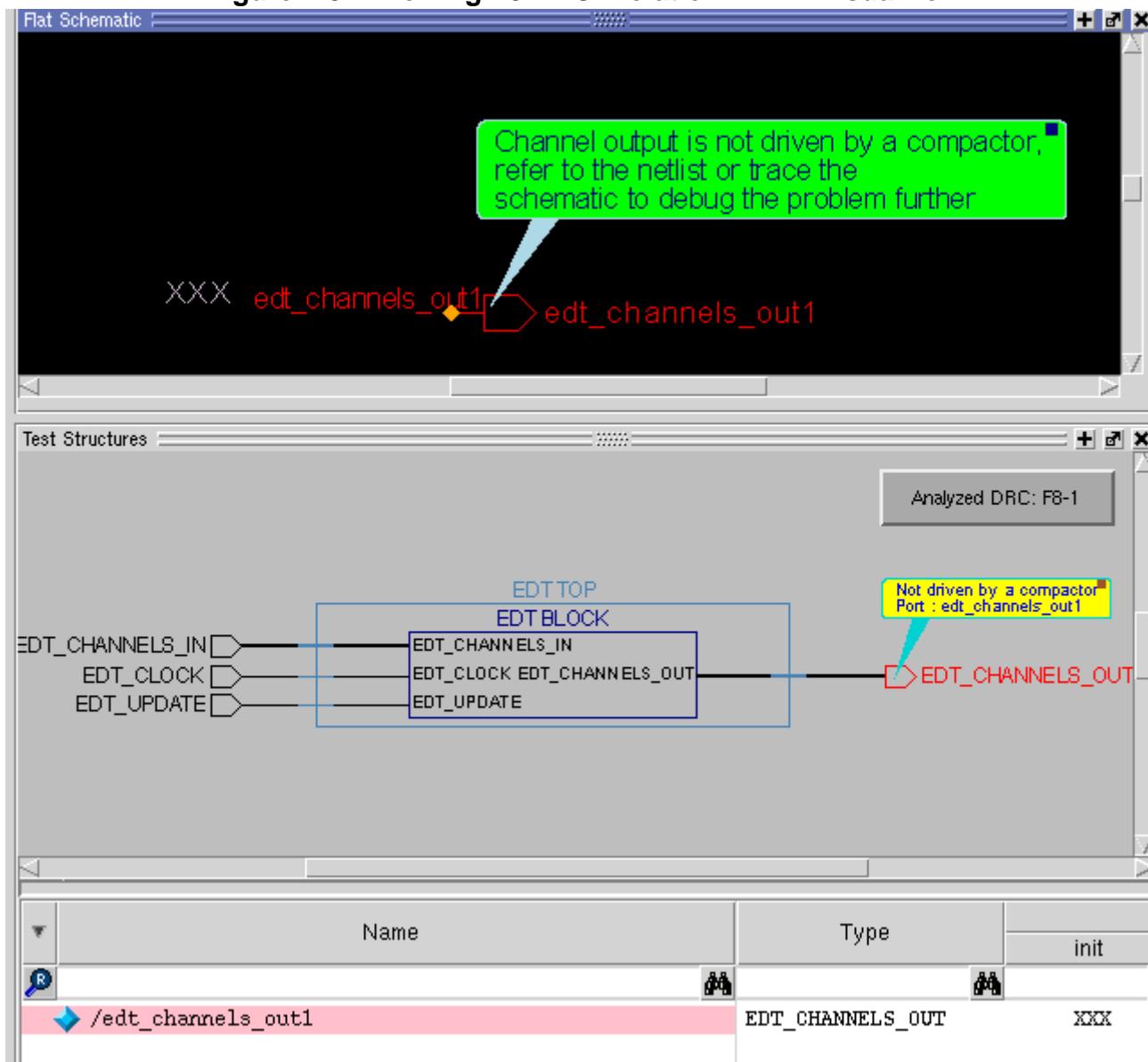
An F8 error can occur when the ATPG model for the pad is incorrect or the pad is incorrectly sensitized. When running in EDT, the F8 failure report may show:

```
// Error: Channel output 1 ('edt_channels_out1') is not driven by a compactor.  
// Either no structural connection exists, or the path is blocked.  
// No state elements reached while tracing back. (F8-1)
```

To view the location of the F8 error, use the following command:

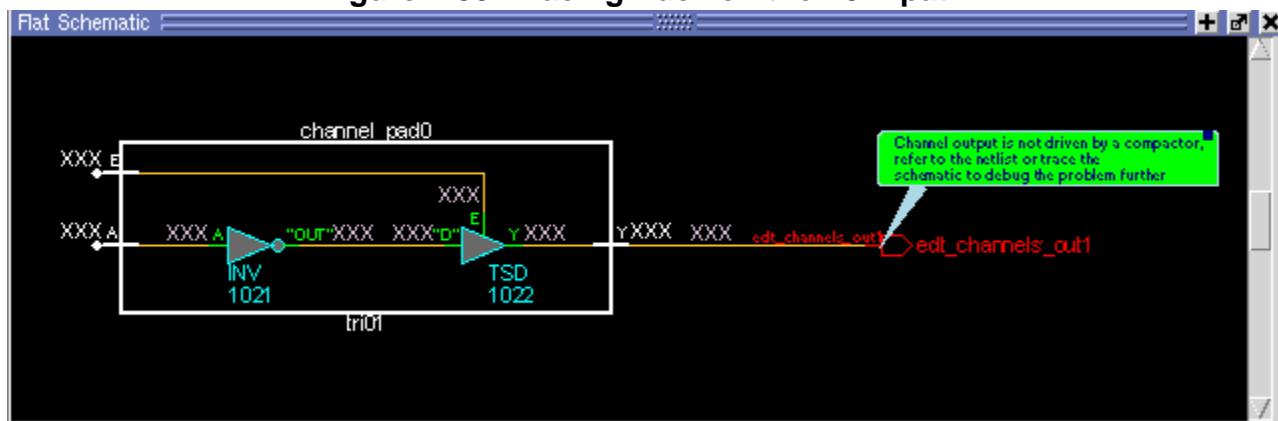
analyze_drcViolation F8-1

This command invokes DFTVisualizer and opens both the Flat Schematic window and the Test Structures window. The Test Structures window highlights the channel output with a callout message of the channel's name as shown in [Figure 7-37](#). The text pane displays the name of the channel, the channel type, and lists the init, capture, load, and shift values. The Analysis field provides a message about the violation.

Figure 7-37. Viewing F8 DRC Violation in DFTVisualizer

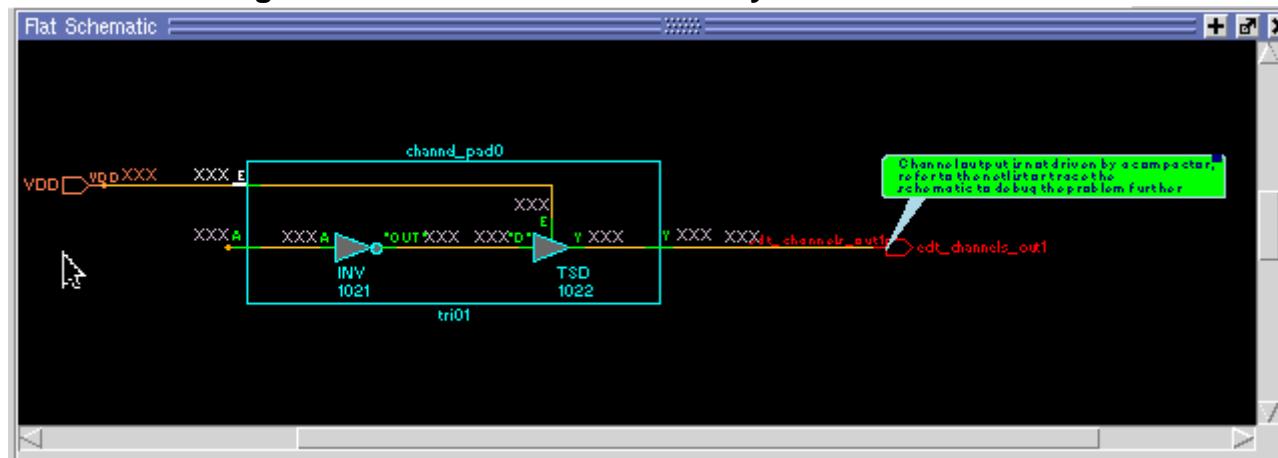
To determine why the data is X, you trace backward from the output channel following the X path. Figure 7-38 shows that `edt_channels_out1` is driven by a tri-state pad.

Figure 7-38. Tracing Back on the F8 X path



Notice that the enable pin of the tri-state device is driven X. As you continue to trace back from the enable pin, Figure 7-39 shows that it is driven by the VDD port.

Figure 7-39. Tri-State Driven to X by connection to VDD



Use this command to solve the problem:

add_input_constraint VDD -CT1

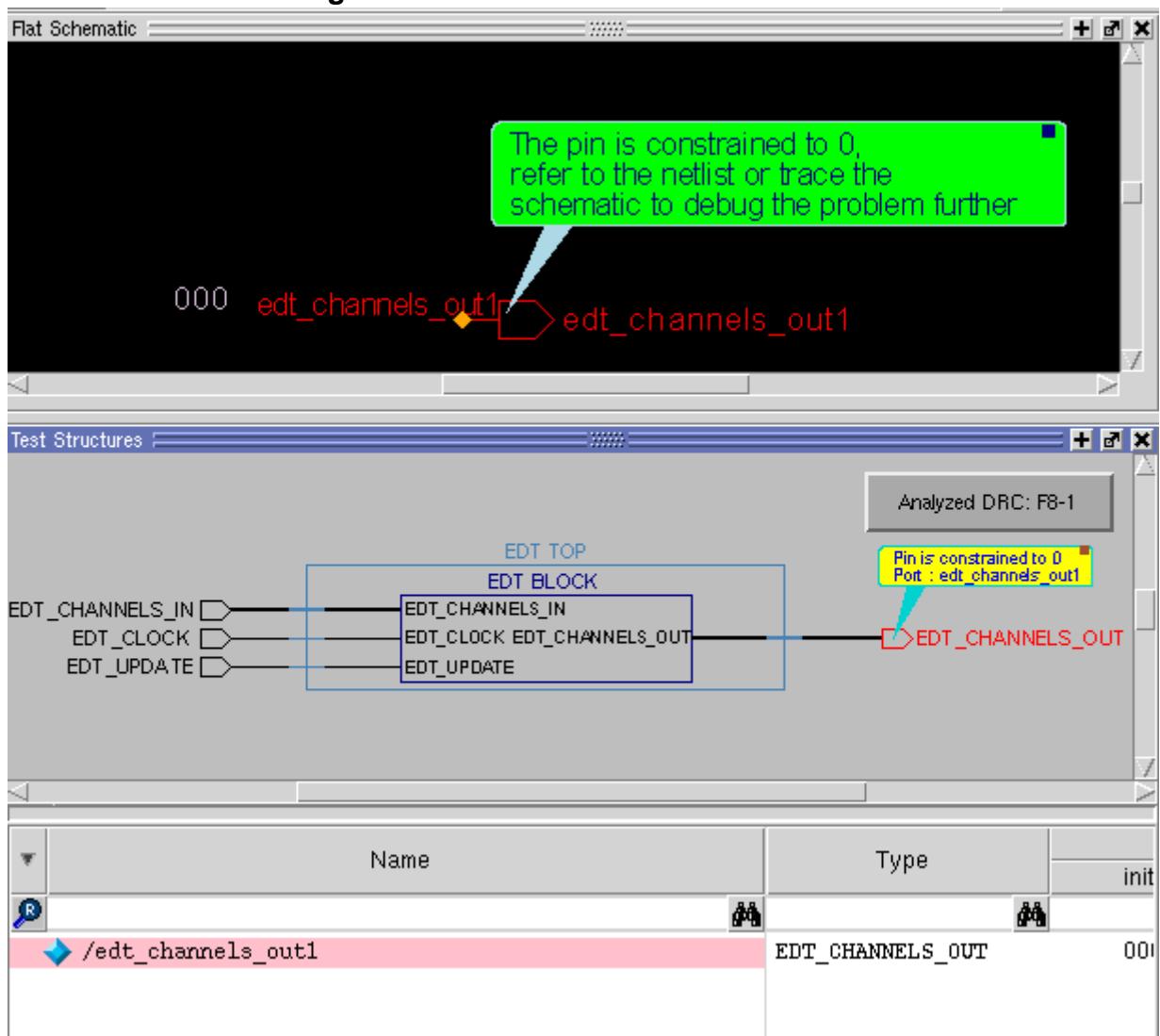
Another possible F8 DRC violation will show:

```
// Error: Channel output 1 ('edt_channels_out1') cannot be traced back
// because it is constrained to 0.
// An unconstrained simulation value (X) is expected. (F8-1)
```

You can use DFTVisualizer to debug it.

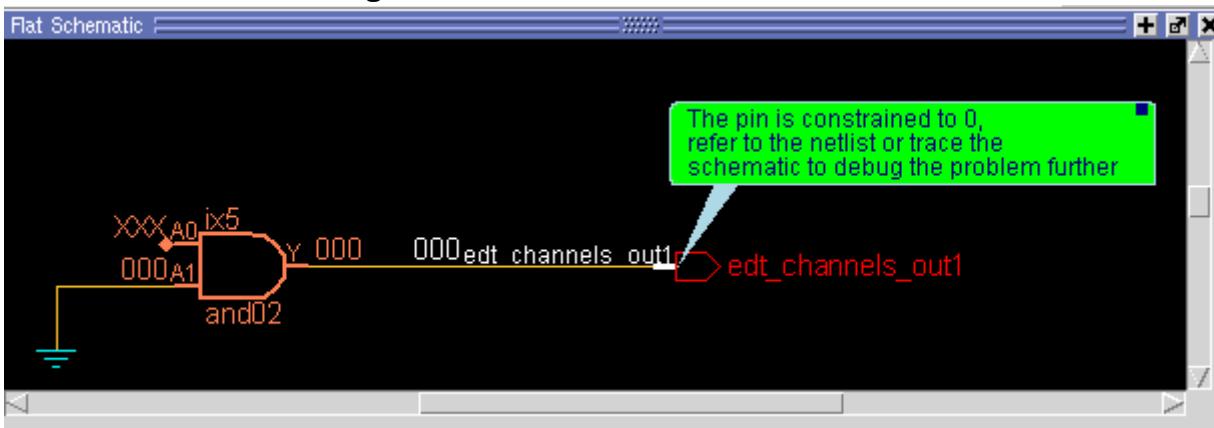
analyze_drcViolation F8-1

Figure 7-40 shows the violation location.

Figure 7-40. F8 DRC Violation - Blocked

To determine why the data is 0, trace back from the `edt_channels_out1` pin. In this case, you will find the output is driven by an AND gate with one pin tied to 0, as shown in Figure 7-41.

Figure 7-41. F8 DRC Violation Cause



To resolve this problem, you must fix the design to ensure that the AND gate driving the output channel does not have one input tied to 0.

F9

Category: EDT

Contexts Supported: patterns -scan

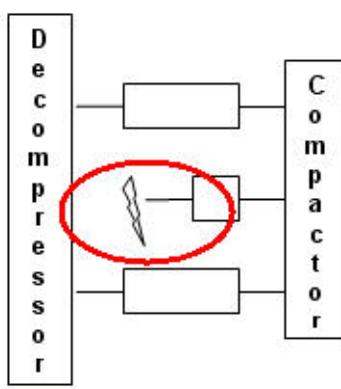
Default Handling: Error

report_drc_rules: Supported

Verifies that all compressed scan chains are connected to a decompressor. F9 traces all channel outputs to verify they connect to a decompressor.

Figure 7-42 shows an error condition.

Figure 7-42. Blocked Scan Chain Path



The error message is:

Compressed scan chain driving channel output P (N) [of EDT block E] blocked after tracing C cells. The last gate reached while tracing back is: M (G). (F9-1)

Where P is the channel index and N is the pin name, E is the name of an EDT block, C is the number of scan cells traced in the scan chain, M is the instance name of the last traced gate and G is its gate ID number.

F10

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

This DRC verifies that the EDT logic found matches the EDT logic expected by the tool.

The EDT logic verification steps include, for example, checking the number of found scan chains compared to the number specified in the dofile, comparing the identified decompressor input taps with the expected ones, and checking if the channel pins driving/connected to a decompressor are consistent with the dofile specifications.

These are examples of the error messages for this case:

The size I1 of decompressor D [of EDT block E] does not match the expected size I2. (F10-1)

The input taps of decompressor D [of EDT block E] do not match the ones expected by the tool. For details about the found decompressor use 'report_edt_finder -Decompressors -Id D -Verbose'. (F10-2)

The expected decompressor taps of scan chain S [of EDT block E] do not match the taps of any found scan chain. (F10-3)

The input taps of channel input N (P) do not match the ones expected by the tool. Verify that the channel inputs are defined in the correct order. (F10-4)

The found number of scan chains (8) driven by the decompressor does not match the expected number (7). (F10-1)

The scan chain 'chain3' could not be found because it does not drive a channel output. Check if its scan output 'ff6/Q' (268) is properly connected to the compactor. (F10-2)

Decompressor 3 is driven by a channel input defined for EDT blocks 'blk1' and 'blk2' ('core2_5_edt_channels_in1'), but is connected to a channel output defined for EDT block 'blk3' ('top_out_7'). Please double-check the EDT channel pin specifications in the dofile. (F10-1)

Where D is the name of the decompressor, E is the name of an EDT block, S is the name of the scan chain, N is the index number of the channel input, and P is the pin name.

This DRC also verifies that the status of the low-power hardware is consistent with the user-specification of the set_edt_power_controller command in the dofile. If the tool identifies low-power hardware but the set_edt_power_controller command in the dofile disables it, the tool issues an error.

If the tool is unable to identify any low-power hardware but the set_edt_power_controller command in the dofile enables it, the tool also issues an error.

Low power hardware was found but should be disabled according to the dofile. Verify that the hardware is disabled in the test procedures.
(F10-1)

The defined low power hardware of EDT block 'blk2' could not be found. Verify that the hardware is enabled in the test procedures. (F10-1)

Examples

This DRC also identifies channel pin definition mismatches. These are examples of the error messages for the case:

```
// Error: Decompressor 2 is connected to channel outputs defined for EDT
// blocks 'm2_4x32' ('m2_edt_channels_out3', 'm2_edt_channels_out4')
// and 'm3_50x187' ('m2_edt_channels_out1', 'm2_edt_channels_out2').
// It is driven by channel inputs defined for EDT block 'm2_4x32'.
// Please double-check the EDT channel pin specifications in the
// dofile. (F10-1)

// Error: Decompressor 2 is driven by channel inputs defined for EDT
// blocks 'm2_4x32' ('m2_edt_channels_in2', 'm2_edt_channels_in3',
// 'm2_edt_channels_in4') and 'm3_50x187' ('m2_edt_channels_in1').
// It is connected to channel outputs defined for EDT block 'm2_4x32'.
// Please double-check the EDT channel pin specifications in the
// dofile. (F10-1)

// Error: Decompressor 3 is driven by channel inputs defined for EDT block
// 'm3_50x187', but is connected to channel outputs defined for EDT
// block 'm2_4x32'.
// One of the channel inputs is 'm3_pdata[0]' and one of the channel
// outputs is 'm3_paddr[0]'.
// Please double-check the EDT channel pin specifications in the
// dofile. (F10-1)

// Error: Decompressor 3 is driven by channel inputs defined for EDT
// blocks 'm2_4x32' ('m3_pdata[0]')
// and 'm3_50x187' ('m3_pdata[1]', 'm3_pdata[2]', 'm3_pdata[3]').
// It is connected to channel outputs defined for EDT blocks
// 'm1_28x16' ('m3_paddr[2]', 'm3_paddr[3]') and 'm3_50x187'
// ('m3_paddr[0]', 'm3_paddr[1]').
// Please double-check the EDT channel pin specifications in the
// dofile. (F10-3)
```

F11

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Ignore

report_drc_rules: Supported

Verifies that all decompressors connect to at least one scan chain and that the path from the decompressor to the scan chains is not blocked due to wrong or missing shift procedure settings.

The default handling for this DRC is “ignore”.

No correction is required if the decompressor is not needed for testing.

The occurrence message is:

Decompressor D has been found but is not used in the current mode. It contains S state elements, one of which is N (G). (F11-1)

Where D is the internal identification number of the decompressor and S is the number of its state elements, N is the instance name of a decompressor state element and G is the corresponding gate ID number.

F12

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

Verifies that all uncompressed scan chains are defined with the add_scan_chains command.

The warning message is:

An uncompressed scan chain has been found that was not defined with the "add scan chain" command. Its scan input is P1, scan output is P2, and it contains S state elements. (F12-1)

Where P1 is a primary input pin, P2 is a primary output pin and S is the number of state elements in the scan chain.

F13

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the decompressor, Xpress mask shift register state elements, and low-power shift registers are reset correctly prior to shift.

The error message in the case of the decompressor is:

The decompressor [of EDT block E] is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing decompressor state elements is: M (G). (F13-1)

In the case of an Xpress compactor, the state elements of the mask shift registers are checked in the same way. The error message is:

The mask shift register driven by channel input P (N) [of EDT block E] is not reset correctly prior to shift. One of the failing mask shift register state elements is: M (G). (F13-1)

In the case of a low-power shift register, the error message is:

The low power shift register driven by channel input P (N) [of EDT block E] is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing low power shift register state elements is: M (G). (F13-1)

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number, and V1 and V2 are gate values.

F14

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask shift registers and the low-power shift registers hold their value during capture.

Only one violation per EDT block is reported.

The error message for mask shift registers (Xpress compactor) is:

The mask shift register driven by channel input P (N) [of EDT block E] does not hold its value during capture. One of the failing mask/power shift register state elements is: M (G). (F14-1)

The error message for mask shift registers (basic compactor) is:

The mask shift register [of EDT block E] does not hold its value during capture. One of the failing mask shift register state elements is: M (G) . (F14-1)

The error message for a low-power shift register is:

The low power shift register driven by channel input P (N) [of EDT block E] does not hold its value during capture. One of the failing low power shift register state elements is: M (G) . (F14-1)

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element and G its gate ID number.

F15

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that input pipeline state elements are initialized correctly. F15 verifies that any state elements that exist between a channel input and the EDT logic are initialized to the right value so that 0s get propagated into the EDT logic during the first shift.

Only one violation per EDT block is reported.

You cannot change the handling with the set_drc_handling command.

The error message is:

State elements in the input pipeline driven by channel input P (N) [of EDT block E] are initialized to wrong values after test_setup and load_unload. One of the failing input pipeline state elements is: M (G) . Simulated value: V1. Expected value: V2. (F15-1)

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, G is the gate ID number, and V1 and V2 are gate values.

F16

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Input pipeline state elements must preserve their value during capture and load_unload. F16 verifies that any state elements between a channel input and the EDT logic preserve the value shifted into them during capture and load_unload procedure.

For additional information on pipelining, see “[Input Channel Pipelines Must Hold Their Value During Capture](#)” in the *Tessent TestKompress User’s Manual*.

Only one violation per EDT block is reported.

The error message is:

```
State elements in the input pipeline driven by channel input P (N)
[of EDT block E] do not preserve their value during capture and
load_unload. One of the failing state elements is: M (G). (F16-1)
```

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number.

F17

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers and the low-power hold registers are clocked during load_unload.

Only one violation per EDT block is reported.

The error message reported when mask hold registers violate this rule is:

```
Mask hold register state elements [of EDT block E] are not clocked during
load_unload. One of the failing mask hold register state elements is:
M (G). (F17-1)
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] are not clocked
during load_unload. One of the failing low power hold register state
elements is: M (G).
(F17-1)
```

In these messages, E is the name of the EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F18

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers load their values from the state elements of the mask shift registers before shift in the load_unload procedure; verifies that the state elements of the low-power hold registers load values from the low-power shift registers before shift in the load_unload procedure

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

```
Mask hold register state elements [of EDT block E] do not load values from
mask shift registers before shift in the load_unload procedure.
One of the failing mask hold register state elements is: M1 (G1).
The driving mask shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2.
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] do not load values
from low power shift registers before shift in the load_unload procedure.
One of the failing low power hold register state elements is: M1 (G1).
The driving low power shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2. (F18-1)
```

In these messages, E is the name of the EDT block, M1 is the name of the failing state element, G1 is its gate ID number, M2 is the name of the driving state element of the low-power hold register and G2 its gate ID number, and V1 and V2 are gate values.

F19

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers and low-power hold registers preserve their values during shift.

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

Mask hold register state elements [of EDT block E] do not preserve their values during shift. One of the failing mask hold register state elements is: M (G) . (F19-1)

The error message reported for low-power hold registers is:

Low power hold register state elements do not preserve their values during shift. One of the failing low power hold register state elements is: M (G) . (F19-1)

In these messages, E is the name of an EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F20

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

The F20 design rule check verifies the masking modes of a compactor.

F20 programs the compactor to the various masking modes by directly loading the mask hold registers and tracing backward from the channel outputs (or from the first pipeline stage in case of a pipelined channel output) to verify that only the expected scan chains are driving the channel outputs. F20 also checks for masked chains with incorrect unload specification.

If the F20 DRC has verified the masking modes (that is, if the check is not set to “ignore”), the tool skips the corresponding K22 DRCs.

Mask-all Masking Mode

If the compactor supports the mask-all masking mode, F20 verifies that all scan chains are masked in that mode; that is, the channel output receives a “0” (or a “1” in the case of an inversion). In the case of a violation, the error message is:

Mask-all masking mode: The channel output P (N) [of EDT block E] receives an X although all scan chains driving it are expected to be masked. The last reached gate while tracing back is: M (G) [which is the scan output of scan chain 'S'] . (F20-1)

Where P is the channel index and N is the pin name, E is the name of an EDT block, M is the instance name of the last traced gate and G is its gate ID number, and S is the name of the found

scan chain. Note, the string “[which is the scan output of scan chain ‘S’]” is only reported if M is the scan output of a scan chain.

1-Hot Masking Mode

F20 verifies that only the expected scan chain drives the expected channel output. All other scan chains are masked. If the expected channel output is not driven by the expected scan chain the error message is:

```
1-hot masking mode for scan chain 'S1':  
The channel output P (N) [of EDT block E] is not driven by gate M1 (G1)  
which is the scan output of scan chain 'S1'.  
The last reached gate while tracing back is:  
M2 (G2) [which is the scan output of scan chain 'S2']. (F20-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, S1 is the name of the expected scan chain, M1 is the instance name of the scan output gate of scan chain S1 and G1 is its gate ID number, M2 is the instance name of the last traced gate and G2 is its gate ID number, and S2 is the name of the found scan chain. Note, the string “[which is the scan output of scan chain ‘S2’]” is only reported if M is the scan output of a scan chain.

F20 checks for masked chains with incorrect unload specification and produces an error message similar to the following:

```
// Finding internal scan chains.  
// Error: The found number of scan chains (2) driven by the decompressor  
// does not match the expected number (1). (F10-1)  
// Error: 1-hot masking mode for scan chain 'chain1':  
//     The channel output 1 (/edt_channels_out1) is driven by gate  
//     '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan  
//     output of chain 'chain1' but the scan chain is defined to be  
//     masked and to unload a constant 0.  
//     The last gate reached while tracing back is:  
//     '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan  
//     output of chain 'chain1' (F20-1)  
  
// Error: XOR masking mode 2: The channel output 1 (/edt_channels_out1)  
// is not driven by the expected scan chains.  
//     The following unexpected driver has been found:  
//     '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan  
//     output of chain 'chain1' (F20-2)
```

Xpress Masking Mode

If the compactor supports the Xpress masking modes, F20 verifies that only the expected scan chains drive the expected channel outputs. All other scan chains are masked. If the expected channel output is not driven by the expected scan chains, the error message is:

```
XOR masking mode K: The channel output P (N) [of EDT block E] is not driven
by the following scan chains:
S1, S2, ..., SN as expected.
The last reached gate while tracing back is:
M (G) [which is the scan output of scan chain 'S']. (F20-1)
```

Where K is the index of the masking mode, P is the channel index and N is the pin name, E is the name of an EDT block, S1, S2, ..., SN are the names of the expected scan chains, M is the instance name of the last traced gate and G is its gate ID number, and S is the name of the found scan chain. Note, the string [which is the scan output of scan chain 'S'] is only reported if M is the scan output of a scan chain.

F21

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

The F21 design rule check verifies that all scan chains driving a compactor have the expected latency; that is, the expected number of pipeline stages between the scan output and the channel output.

In some cases, EDT Finder learns and adjusts the number of compactor and output channel pipeline stages to match what was found in the design. If it succeeds in learning and adjusting the number of pipeline stages, the F21 DRC is not issued.

EDT Finder only adjusts the number of compactor pipeline stages if the number of compactor pipeline stages in the EDT IP was not passed from the IP Creation phase. Normally this information is passed through and any discrepancy will result in a DRC violation.

In the case of a violation, the error message is:

```
Incorrect number of pipeline stages between the scan chain 'S'
and the channel output P ('N') [of EDT block 'E']: identified=N1
expected=N2.
The scan output of scan chain 'S' is M (G)
The following pipeline stages have been identified:
P1 (G1)
P2 (G2)
...
PN (GN) (F21-1)
```

Where S is the name of the scan chain, P is the channel index and N is the pin name, E is the name of an EDT block, N1 and N2 are the number of identified and expected pipeline stages, M is the scan output of the scan chain and G is its gate ID number, and P1, P2, ..., PN are the identified pipeline stages and G1, G2, ..., GN their gate ID numbers.

F22

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

When two adjacent memory elements (source and sink) in the scan path are clocked by different shift clocks, the sink must not capture data from the source at the same time that the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan shifting when clock skew exists between the different shift clocks.

This DRC verifies this rule for EDT-related state elements. This DRC complements the T24 DRC that verifies this rule for the scan chains. Together, these two DRCs cover the complete scan path.

In the case of a violation, the error message is:

Two adjacent pipeline cells of channel input P (N) [of EDT block E]
are clocked on the same edge by different clocks.

N1 (G1), driven by clock N3 (G3), and
N2 (G2), driven by clock N4 (G4)

Consider adding a lockup cell between them, or modifying the clock timing.
(F22-1)

The first sentence of the error message depends on the location of the violation. Other possibilities are:

Two adjacent cells of the pipeline of channel input P (N) and the decompressor [of EDT block E] ...

Two adjacent cells of the decompressor [of EDT block E] and scan chain 'C1' ...

Two adjacent cells of scan chain 'C1' and the compactor [of EDT block E] ...

Two adjacent cells of scan chain 'C1' and the pipeline of channel output P (N) [of EDT block E] ...

Two adjacent cells of the compactor [of EDT block E] ...

Two adjacent cells of the compactor and the pipeline of channel output P (N) [of EDT block E] ...

Two adjacent pipeline cells of channel output P (N) [of EDT block E] ...

N1 and N2 are the instance names of the adjacent elements clocked by different clocks, N3 and N4 are the instance names of the clocks, G1-G4 are the corresponding gate IDs, P is the channel index and N the pin name, E is the name of an EDT block, and C1 is the name of a scan chain.

Note

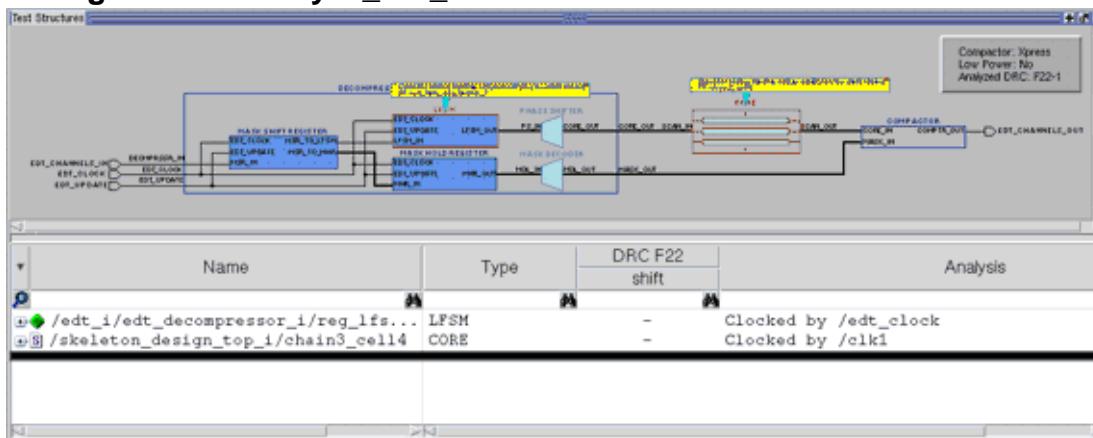
 If the F22 DRC has verified this rule (that is, the set_drc_handling command is not set to "ignore" F22 violations) the K21 DRC is skipped. This only applies to the pattern generation phase and not the IP creation phase.

How to Debug Using DFTVisualizer

To view the location of a F22 DRC violation using DFTVisualizer, use the following command:

analyze_drcViolation F22-1

This command invokes DFTVisualizer and opens both the Flat Schematic window and the Test Structures window. The Test Structures window shows a representation of the test structure and highlights the location of the two adjacent cells with a callout message of the cell's name as shown in [Figure 7-43](#) on page 2819. The text pane displays the name of the cell, the type of cell, and, in the Analysis field, lists the controlling clock for that cell.

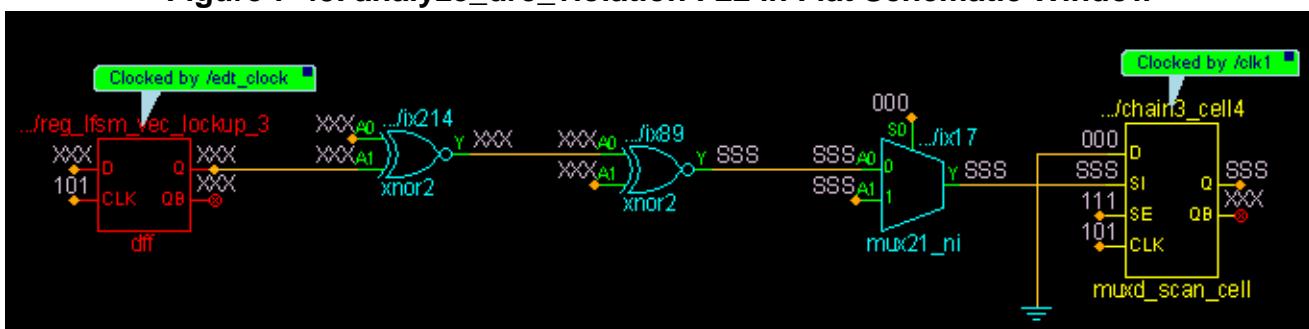
Figure 7-43. analyze_drcViolation F22 in Test Structures Window


In this example, the Test Structures window callouts indicate that the F22 violation occurs because a lockup cell is missing in the path from the decompressor to the scan chain. As shown in [Figure 7-44](#), this typically happens if you use the skeleton flow and define incorrect clock timing for the first cell of the scan chain.

Figure 7-44. Callout Text for reg_lfsm_vec_lockup_3

Consider adding a lockup cell, or modifying the clock timing
 FF : reg_lfsm_vec_lockup_3

The Flat Schematic window shows the connected path between the two adjacent cells and displays in the callouts the two different clocks that pulse on the same edge. The gate report is set to the shift procedure so you can see that both cells capture on the same edge.

Figure 7-45. analyze_drcViolation F22 in Flat Schematic Window


This type of violation typically occurs because during IP creation the clock of the first scan cell in the chain is leading edge (010), but after synthesis the first cell is actually a trailing edge flop (101). When you are using the skeleton flow, it is important that the edge of the first and last cell is defined correctly or the tool may end up with an extra or missing lockup cell.

Solution

You can correct this violation by inserting a lockup cell between the adjacent memory elements reported in the violation message, or by regenerating the EDT IP using the correct edges for the first cell.

Flattening Rules (FN, FP, and FG Rules)

The flattening rules check the hierarchical design's suitability for flattening.

The flattening rule violations and their identification literals are divided into the following three groups:

- Net rules — FN1 through FN9
- Pin rules — FP1 through FP13.
- Gate rules — FG1 through FG8.

You can use the `set_flattener_rule_handling` command to change the handling of the net, pin, and gate flattening rules.

FN1	2822
FN2	2822
FN3	2822
FN4	2822
FN5	2823
FN6	2823
FN7	2823
FN8	2823
FN9	2824
FP1	2824
FP2	2824
FP3	2824
FP4	2824
FP5	2825
FP6	2825
FP7	2825
FP8	2825
FP9	2826
FP10	2826
FP11	2826
FP12	2826
FP13	2826
FG1	2827
FG2	2827

FG3	2827
FG4	2827
FG5	2828
FG6	2828
FG7	2828
FG8	2828

FN1

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a module net is floating. By default, the tool ignores this rule in the RTL context. You can overwrite this default handling by using the set_flattener_rule_handling command.

FN2

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a module net has a driver and a constant value property. The constant value property is ignored.

FN3

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning for non-RTL contexts, Ignore for RTL contexts. (You can overwrite this default handling by using the set_flattener_rule_handling command.)

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an instance net is floating.

FN4

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an instance net is not used.

FN5

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule identifies a wired net with multiple drivers.

FN6

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a bus net attribute cannot be used.

FN7

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if two connected nets have inconsistent net attributes. Both inconsistent attributes are ignored.

FN8

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule identifies parallel wired behavior.

FN9

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the bus net has multiple bus keepers. The effects of the bus keepers are additive.

FP1

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the circuit has no primary inputs.

FP2

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the circuit has no primary outputs.

FP3

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a primary input drives logic gates and switch gates.

FP4

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a pin was moved.

FP5

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a pin was deleted by merging.

FP6

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule identifies merged wired in/out pins.

FP7

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule identifies merged wired input and output pins.

FP8

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if a module boundary pin has no name.

FP9

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an inout pin is used as output only.

FP10

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use [report_flattener_rules](#)

This rule determines if an output pin is used as an inout pin.

FP11

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an input pin is used as an inout pin.

FP12

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an output pin has no fan-out.

FP13

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if an input pin is floating. By default, the tool ignores this rule in the RTL context. You can overwrite this default handling by using the `set_flattener_rule_handling` command.

FG1

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

`report_drc_rules`: Not Supported. Use [report_flattener_rules](#).

This rule determines if the defining model of an instance does not exist. If you reduce the handling to warning or note, this primitive is treated as undefined.

FG2

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

`report_drc_rules`: Not Supported. Use [report_flattener_rules](#).

This rule determines if the feedback gate is not in feedback loop.

FG3

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

`report_drc_rules`: Not Supported. Use [report_flattener_rules](#).

This rule determines if the bus keeper has no functional impact.

FG4

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

`report_drc_rules`: Not Supported. Use [report_flattener_rules](#).

This rule determines if the RAM/ROM read attribute is not supported.

FG5

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the RAM attribute not supported.

FG6

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the RAM type is not supported.

FG7

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the netlist module has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

FG8

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use [report_flattener_rules](#).

This rule determines if the library model has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

General Rules (G Rules)

G rules are general rules that identify inconsistencies in scan data and other definitions. G rules are the first rules checked during the DRC process and all violations generate error conditions that must be corrected. You cannot change the handling of these rules.

G1	2829
G2	2829
G3	2830
G4	2830
G5	2830
G6	2831
G7	2831
G8	2831
G9	2832
G10	2832
G11	2833
G12	2833

G1

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Each defined scan chain group, except for “dummy,” must contain at least one scan chain. You can correct this error condition by either adding a scan chain to the group or by deleting the scan chain group.

The error message is:

```
No scan chains have been defined for group N. (G1-1)
```

N is the name of the scan chain group, and G1-1 indicates the rule and violation ID numbers.

G2

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define scan chains and do not use the “dummy” scan chain option, you must define at least one clock. You can correct this error condition by either defining a clock that controls the defined scan chains or by deleting all scan chain groups.

The error message is:

```
Scan chains exist but no synchronous clocks have been defined. (G2-1)
```

G3

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If the circuit has no memory elements, you cannot define clocks. You can correct this error condition by deleting all clocks.

The error message is:

```
Clocks are defined but no memory elements exist in the circuit. (G3-1)
```

G4

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If the circuit has no memory elements, you cannot define scan chain groups. You can correct this error condition by deleting all scan chain groups.

The error message is:

```
Scan groups are defined but no memory elements exist in the circuit. (G4-1)
```

G5

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If there are no RAMs in the circuit, you cannot define write control lines. You can correct this error condition by deleting all write control lines.

The error message is:

```
Write controls are defined but no RAMs exist in the circuit. (G5-1)
```

G6

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define a linear feedback shift register (LFSR), you cannot use the “dummy” scan chain option. You can correct this error condition by either deleting all LFSRs or deleting the dummy scan chain group.

The error message is:

```
Cannot use dummy scan chain with BIST LFSRs. (G6-1)
```

G7

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

The RAM/ROM instance name given on a preceding read_modelfile command must contain a single RAM or ROM gate. You can correct this error condition by using the correct RAM or ROM instance name for the read_modelfile command. Or, you can do nothing and re-invoke the rules checker, in which case, the tool will not use a modelfile for the intended RAM or ROM.

The error message is:

```
Cannot use RAM/ROM modelfile M for invalid instance N. (G7-1)
```

M is the modelfile name, N is the instance name, and G7-1 indicates the rule and violation ID numbers.

G8

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

All ROM gates must have a defined initialization file, unless you use the random initialization option. You can correct this error condition by: specifying a modelfile in the model cell library, using the read_modelfile command to specify a modelfile, using random initialization, or changing the model cell library to treat the ROM gate as undefined.

The error message is:

```
ROM initialization file not defined for N (G) . (G8-1)
```

N is the instance name of the ROM, G is the gate ID number, and G8-1 indicates the rule and violation ID numbers.

G9

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

For all constrained scan cells identified by chain and position, the scan chain must be a valid scan chain, the position must be less than the length of the chain, and the scan cell must not be the same as another constrained scan cell. You can correct this error by identifying and correcting all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint position P for chain C. (G9-1)
```

P is the cell position number (0-based, where 0 is the scan cell closest to the scan-out pin), C is the scan chain name, and G9-1 indicates the rule and violation ID numbers.

G10

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

For all constrained scan cells identified by pin pathname, the pin must be a valid output pin of a cell, the pin must connect to a scan memory element through a path that only contains buffers and inverters, and the scan cell must not be the same as another constrained scan cell. To correct this error, you must identify and correct all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint pin name P. (G10-1)
```

P is the pin pathname of an output pin of a cell, and G10-1 indicates the rule and violation ID numbers.

G11

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define a “dummy” scan chain group with a test procedure file, you cannot define any scan chains. The purpose of the dummy scan group is to provide the ability to use a **test_setup** procedure when no scan cells exist. To correct this error (if scan cells do exist), you should place the **test_setup** procedure in the test procedure file for a defined scan chain group.

The error message is:

```
Scan chains may not be defined when using dummy scan group procedure file.  
(G11-1)
```

G12

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

General rule 12 reports add_cone_blocks command pin-pathname violations because their checking is delayed until DRC is performed.

ICL Extraction Rules (I Rules)

Tessent Shell currently performs limited ICL Extraction rule checks on the design.

Table 7-12 lists the ICL-specific DRC rules that check the mandatory pre-conditions the design must have in order to extract the ICL top level from the netlist.

Table 7-12. ICL Extraction DRCs

DRC	Description
I1	Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command. Also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module.
I2	Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port. Also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL.
I3	Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.
I4	This rule checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.
I5	Checks the consistency of specified pin constraints on ICL-attributed design instances.
I6	Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.

The ICL-specific DRC rules described in this section are based on the [IEEE 1687-2014](#) (IJTAG) standard. These checks are only performed in the Tessent Shell patterns -ijtag context—see “[ICL Extraction](#)” in the *Tessent IJTAG User’s Manual*.

Refer to “[Design Rule Checks](#)” in the *Tessent IJTAG User’s Manual* for complete information.

I1

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command.

Specifically, the tool verifies that all ports on the mapped ICL module are present in the corresponding design module. If an ICL module port is not found in the corresponding design module, the tool issues the following error messages:

The port 'P' exists in the description of the ICL module 'M1' but not on the matching design module 'M2'.

Where:

'P' is the name of the ICL module port, 'M1' is the name of the ICL module, and 'M2' is the name of the design module matching the ICL module 'M1'.

The tool also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module. If a design module port has a conflicting direction, the tool issues the following error messages:

DataInPort 'PI' exists in the description of the ICL module 'M1'. However, the corresponding port in the design module 'M2' is not of a compatible direction. An ICL port of type DataInPort must map to a design port that is either an input or bidirectional port.

DataOutPort 'PO' exists in the description of the ICL module 'M1'. However, the corresponding port in the design module 'M2' is not of a compatible direction. An ICL port of type DataOutPort must map to a design port that is either an output or bidirectional port.

Where:

'PI' and 'PO' are the names of the ICL module ports, 'M1' is the name of the ICL module, and 'M2' is the name of the design module matching the ICL module 'M1'.

I2

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port.

This DRC violation typically occurs when a pin is unconnected. Specifically, this rule checks that every ICL-attributed instance input pin is connected to either an ICL-attributed instance

output pin or a primary input port, and every ICL-attributed instance output pin is connected to either an ICL-attributed instance input pin or a primary output port.

The tool also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL using the allowed_tied_high or allowed_tied_low value for the connection_rule_option attribute, and reports an I2 violation if there is a mismatch. If the simulated value is X and the connection_rule_option is not allowed_no_source then the failure to drive the input pin is reported.

These checks are performed for all ICL input ports and for ICL output ports with the port functions DataOutPort and ScanOutPort.

If a connection cannot be identified, the tool issues the following error message:

```
The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
```

If the trace stopped at only one gate, which is always the case in the backward trace, the tool issues one of the following error messages which includes the gate information:

```
The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate '/I/Q'.
```

```
The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The forward trace stopped at gate '/I/Q'.
```

If there is a mismatch between the simulated value and the tied value, the tool issues one of the following error messages:

```
Mismatch between permissible tied value (1) declared by attribute
'connection_rule_option' for DataInPort 'P' on ICL-attributed design
instance 'I', module 'M' and the observed simulated value 0.
```

```
Mismatch between permissible tied value (0) declared by attribute
'connection_rule_option' for DataInPort 'P' on ICL-attributed design
instance 'I', module 'M' and the observed simulated value 1.
```

Where:

'P' is the name of the DataInPort, 'Q' is the name of the DataOutPort, 'I' is the instance name of the ICL-attributed start instance, 'M' is the module name of the ICL-attributed start instance. The DFTVisualizer can then highlight the starting instance 'I' and pin 'P' and the last gates that was reached during the tracing process.

I3

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.

If the port functions do not match, the tool issues the following error message:

```
The connection between ScanOutPort 'SO' on ICL-attributed design instance
'I1', module 'M1' and ClockPort 'SI' on ICL-attributed design instance
'I2', module 'M2' violates the ICL port function connection rules.
```

```
An instance ClockPort is only allowed to be driven by the following
sources: ClockPort on the parent module or ToClockPort reference from an
instance within the parent module. Where recognized values of the
function_modifier attribute are used to alter ICL port semantics they
must be applied consistently.
```

Where:

'I1' and 'I2' are the instance names of the ICL-attributed instances, 'M1' and 'M2' are the module names of the ICL-attributed instances, 'SO' is the name of the ScanOutPort on ICL-attributed design instance I1, 'SI' is the name of the ClockPort on ICL-attributed design instance 'I2'.

The port functions of the top-level module ports are inferred from the identified connections according to [Table 7-13](#) and are attached as port function attributes on the top level module ports.

Table 7-13. Port Connections

Start ICL-attributed Instance Port Function	Inferred Top Module Port Function
scan_in	scan_in
shift_en	shift_en
capture_en	capture_en
update_en	update_en
data_in	data_in
select	select
address	data_in
write_en	data_in

Table 7-13. Port Connections (cont.)

Start ICL-attributed Instance Port Function	Inferred Top Module Port Function
read_en	data_in
scan_out	scan_out
data_out	data_out
to_shift_en	to_shift_en
to_capture_en	to_capture_en
to_update_en	to_update_en
to_select	to_select
to_ir_select	to_select
to_reset	to_reset
to_clock	to_clock

I4

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

This rule checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.

For example, a top-level module input port cannot be both a scan_in signal and a tck clock; it must be one or the other. If the inferred port function of port 'SI' is 'scan_in' due to an identified connection to port 'P1' and the inferred port function of port 'SI' is also 'TCK' due to another identified connection to port 'P1', the tool issues this error message.

```
Inconsistent port functions found for primary port /TCK.
Port function ScanInPort inferred due to a connection started at port 'SI'
on ICL-attributed design instance 'I1', module 'M1' is not consistent
with port function TCKPort inferred due to a connection started at port
'TCK' on ICL-attributed design instance 'I2', module 'M2'.
```

Where:

'SI' is the ICL-attributed start port of the first connection, 'I1' is the instance name of the ICL-attributed start instance of the first connection, 'M1' is the module name of the ICL-attributed start instance of the first connection, 'TCK' is the inferred port function of the primary port due to the second connection, 'I2' is the instance name of the ICL-attributed start instance of the

second connection, and ‘M2’ is the module name of the ICL-attributed start instance of the second connection.

If the polarity inferred due to the connection from the primary input ‘P’ to the module port ‘P1’ is different from the polarity inferred due to the connection from that primary input to another module port ‘P2’, the following error message will be issued:

```
Inconsistent polarity for reset port functions found for primary port P.  
Polarity 'Low'/'High' inferred due to a connection started at port 'P1' on  
ICL-attributed design instance 'I1', module 'M1' is not consistent with  
the polarity 'High'/'Low' inferred due to a connection started at port  
'P2' on ICL-attributed design instance 'I2', module 'M2'.
```

Where:

‘P’ is the primary port name, ‘P1’ is the ICL-attributed start port of the first connection, ‘I1’ is the instance name of the ICL-attributed start design instance of the first connection, ‘M1’ is the module name of ‘I1’, ‘P2’ is the ICL-attributed start port of the second connection, ‘I2’ is the instance name of the ICL-attributed start design instance of the second connection, ‘M2’ is the module name of ‘I2’.

I5

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error

report_drc_rules: Supported

Checks the consistency of specified pin constraints on ICL-attributed design instances.

This DRC checks that the input ports specified in the forced_high_input_port_list are actually simulated to 1 in the stable_after_setup simulation conditions. It also checks that the input ports specified in the forced_low_input_port_list are actually simulated to 0 in the stable_after_setup simulation conditions. You typically need to adjust your test_setup procedure to correct this situation.

The error message is:

```
Input constraint verification failed for input pin 'P' on ICL-attributed  
design instance 'I', module 'M'.
```

```
The corresponding design module port was specified with the attribute 'A'  
in file 'F' on or near line 'L'.
```

```
Expected V1, simulated V2.
```

Where:

'P' is the name of the failing pin, 'I' is the instance name of the ICL-attributed design instance of 'P', 'M' is the module name of 'I', 'A' is the input constraint attribute name (either forced_high_input_port_list or forced_low_input_port_list) defined for 'P' in the ICL file 'F' on line number 'L'. 'V1' and 'V2' are the simulation values on the gate corresponding to the failing pin in the flat netlist.

I6

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.

If no primary input port has been identified as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P)
on ICL-attributed design instance (I) of module (M) because no extracted
primary port with TCK attribute has been identified. (I6-1)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out functionality, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I).

If the enable port of a latch does not have a fixed simulation value and the backward tracing from the enable port does not end at the primary input port that has been identified before as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because it requires an
inverted connection to the extracted primary port (P2) with TCK
attribute. Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate (G). (I6-2)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out ICL attribute, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I), (P2) is the name of the primary input identified as TCK port during extraction and (G) is the name and ID of the last traced gate.

The DFT visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the last gates that were reached during the backward tracing process of the latch enable port.

If the enable port of a latch has a fixed simulated value, the following error message is issued:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because it requires an
inverted connection to the extracted primary port (P2) with TCK
attribute. The path is blocked by a signal setting. (I6-3)
```

Due to the blocking signal settings the DFT visualizer will highlight the gates that were reached during a structural backward tracing process of the latch enable port, the starting instance (LI), the identified primary port TCK (P2) and the scan out pin.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I),

If the enable port of a latch does not have a fixed simulated value and the backward tracing ends at the primary input port that has been identified before as TCK port, but a low signal simulation does not enable the latch, the error message is:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because a low signal at
the extracted primary pin (P2) with TCK attribute will not enable the
latch. (I6-4)
```

The DFT visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the path from the enable port of the latch to the (P2) port.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I),

I7

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

This DRC only applies to design ports which have been subject to the add_clocks command.

These ports are referred to as clocks.

This rule checks if the properties of the clocks are consistent with the port functions and function_modifier attributes of the related extracted ICL ports.

The following is checked in detail:

- If a clock is mapped to an extracted ICL port, then this ICL port must be one of the following:
 - ClockPort
 - TCKPort
 - CaptureEnPort with attribute function_modifier = "CaptureShiftClock"
 - CaptureEnPort with attribute function_modifier = "CaptureShiftClockInv"
- If a clock is mapped to an extracted ICL port of type TCKPort, then it must not be an asynchronous clock.

The following error message is issued in case of a violation against the first requirement:

```
// Error: The DataInPort 'dat' on ICL-attributed design instance 'inst',
// module 'block' is connected to the primary input port 'DAT',
// which is specified as a clock.
// Top module clock ports are allowed to be connected only to
// module ports defined as ClockPort, TCKPort or module ports
// defined as CaptureEnPort when qualified with the ICL attribute
// function_modifier = "CaptureShiftClock" or
// function_modifier = "CaptureShiftClockInv". (I7-1)
```

The following error message is issued in case of a violation against the second requirement:

```
// Error: The ClockPort 'clk' on ICL-attributed design instance 'inst',
// module 'block' is connected to the primary input port 'CLK',
// which is not specified as a pulse-always clock. (I7-3)
```

The following error message is issued in case of a violation against the third requirement:

```
// Error: The TCKPort 'tck' on ICL-attributed design instance 'inst',
// module 'block' is connected to the primary input port 'TCK',
// which is specified as an asynchronous clock. (I7-4)
```

ICL Semantic Rules (ICL Rules)

The tool performs ICL semantic rule checks upon reading (using the `read_icl` command) an ICL file into the tool. These checks provide a more extensive check of the ICL upon reading the file or files into Tessent IJTAG. Basically, the ICL must conform to all of the semantic rules set out for the ICL in the IEEE 1687-2014 (IJTAG) standard.

The semantic rule checks are as follows:

- **Parser** — When you read in an ICL module, the tool parses the file with a series of checks to detect for errors. The tool terminates the ICL parsing checks if any error is detected.
- **Module** — Once the complete module is available, the tool performs an extensive set of semantic checks using the totality of information contained exclusively within the module.
- **Hierarchical Module** — When the `set_current_design` command is issued the entire ICL module instance hierarchy is traversed and each module is rechecked, taking note of any instance parameter overrides on the module instances. Also, at this time those semantic rules that check the relationships between modules, such as module instance port binding semantics, are validated. If any of these checks fail then the ICL hierarchy is not well formed and cannot be used, the `set_current_design` command will fail.
- **ICL Primitive Model** — A further set of checks are performed, using the more detailed ICL primitive model, these are performed at the primitive model bit level and generally check interconnection semantics locally, within a single module, or on the flattened ICL primitive representation, at the global level. Again, if any of these checks fail then the ICL hierarchy is not well formed and cannot be used, the `set_current_design` command will fail.

Note



Many of the ICL Rules have multiple messages associated with them as some rules have broad classes of failure modes

ICL1	2847
ICL2	2848
ICL3	2848
ICL4	2848
ICL5	2849
ICL6	2849
ICL7	2849
ICL8	2849
ICL9	2850
ICL10	2850

ICL11	2850
ICL12	2851
ICL13	2851
ICL14	2851
ICL15	2851
ICL16	2852
ICL17	2852
ICL18	2852
ICL19	2853
ICL20	2853
ICL21	2853
ICL22	2853
ICL23	2854
ICL24	2854
ICL25	2854
ICL26	2854
ICL27	2855
ICL28	2855
ICL29	2855
ICL30	2855
ICL31	2856
ICL32	2856
ICL33	2856
ICL34	2857
ICL35	2857
ICL36	2857
ICL37	2857
ICL38	2858
ICL39	2858
ICL40	2858
ICL41	2859
ICL42	2859
ICL43	2859
ICL44	2860
ICL45	2860

ICL48	2860
ICL49	2860
ICL50	2861
ICL51	2861
ICL52	2862
ICL53	2862
ICL54	2862
ICL55	2863
ICL56	2863
ICL57	2863
ICL58	2863
ICL59	2864
ICL60	2864
ICL61	2864
ICL62	2865
ICL63	2865
ICL64	2865
ICL65	2865
ICL66	2866
ICL67	2866
ICL68	2866
ICL70	2866
ICL71	2867
ICL75	2870
ICL76	2870
ICL77	2870
ICL78	2871
ICL79	2871
ICL80	2872
ICL81	2872
ICL82	2872
ICL83	2873
ICL84	2873
ICL85	2873
ICL86	2873

ICL87	2874
ICL88	2874
ICL89	2874
ICL90	2875
ICL91	2876
ICL92	2876
ICL93	2876
ICL94	2877
ICL95	2877
ICL96	2877
ICL97	2877
ICL98	2878
ICL99	2878
ICL100	2878
ICL101	2879
ICL102	2879
ICL103	2879
ICL104	2880
ICL105	2880
ICL106	2881
ICL107	2881
ICL108	2882
ICL109	2882
ICL110	2882
ICL111	2882
ICL112	2883
ICL113	2883
ICL114	2884
ICL115	2884
ICL116	2884
ICL117	2885
ICL118	2885
ICL119	2885
ICL120	2886
ICL121	2886

ICL122	2886
ICL123	2886
ICL124	2887
ICL125	2887
ICL126	2888
ICL127	2888
ICL128	2888
ICL129	2889
ICL130	2889
ICL131	2890
ICL132	2890
ICL133	2891
ICL134	2891
ICL135	2892
ICL136	2893
ICL137	2893

ICL1

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source files have no parser syntax errors.

A failure of this semantic rule occurs if the input ICL source is not in compliance with the formal grammar for ICL. The accompanying message will show where the tool was when it encountered the incorrect syntax:

```
// Error: Incorrect character sequence. Encountered on or near line 5
// of file source.icl:
//           ^
//           ICL1
```

Note that the initial location of the incorrect syntax may actually be found prior to the location reported.

ICL2

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no un-terminated multi-line comments.

ICL3

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module contains non-overlapping names within the single naming region used for several types of design objects.

This rule checks the following objects:

- Ports
- Registers
- One-hot groups
- Multiplexers
- Alias description
- Logic Signals

ICL4

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module does not contain a duplicate attribute name for a single attributed declaration scope, such as a module or a register.

ICL5

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate RefEnum usage in those contexts where a binding to an enumeration can be made. ICL allows only a single enumeration set to be referenced.

ICL6

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain incorrectly cased ICL keywords. ICL keywords are case sensitive.

ICL uses mixed case keywords, that must be typed exactly as specified. This check will report a failure when the reading of an ICL source file encounters a context that is expecting one of these keywords but sees an incorrectly capitalized version of a valid keyword.

ICL7

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an identifier does not exceed the maximum character length.

The maximum length for an identifier is 1024 characters.

ICL8

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has at least one module.

ICL9

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate ICL module names.

All of the modules within a single ICL source file must be unique. A source file is allowed to provide a new definition of a module that has previously been read from another ICL source file; in this case the previous definition of the module is replaced with the new module definition, if the ICL source file is read without error.

ICL10

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate ICL parameter names within the definition of a single module.

ICL11

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate ScanInterface names within the definition of a single module.

ICL12

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate property specifications.

For example, a DataOutPort declaration may provide a specification of the source value for the port using the Source keyword. This specification may only be made once within the declaration of the port.

ICL13

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that an ActivePolarity specification of a ResetPort or ToResetPort has one of the values 0 or 1.

ICL14

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate instance names within a module.

ICL15

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the ICL source file contains no directly self-referential instances.

A module is not allowed to instance itself. Circular instance hierarchies are also not allowed, and are caught by another rule check.

ICL16

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that each module instance within a one hot data group declares the address value for that instance, using the AddressValue specification.

ICL17

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that within a module definition, contained in an ICL source file, each module instance does not have multiple overrides for the same instance parameter name.

ICL18

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the specification of an address value for a module instance only occurs within module instances that are defined within a one hot data group. Module instances that occur outside of a one hot data group, in the main body of a module, cannot have address value declarations, using the AddressValue specification.

ICL19

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Each scan register declaration must define the source of the scan input, using the ScanInSource specification.

ICL20

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file contains only supported ICL Constructs.

Tessent IJTAG does not yet provide support for the full ICL syntax. This rule will fail when the ICL reader encounters a piece of ICL syntax that is not currently supported.

ICL21

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the data register declaration within a one hot data group provides a definition of the address value for that data register, using the AddressValue specification.

ICL22

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a data register declaration that contains no write data source, which implies that the register is read only, also does not provide a write enable source. A read only data register does not require and must not have a write enable control.

ICL23

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a data register with an address value definition does not also have a write enable definition. Where a data register is defined within a one hot data group and has an address value the address and explicit write enable control will be used to control write accesses to the data register. Such an addressed data register does not require and must not define a write enable source.

ICL24

Reserved for future use.

ICL25

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an addressed data register does not have a write data source.

An addressed data register will implicitly obtain the write data source from the data inputs of the module.

ICL26

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate enumeration declarations. Within an ICL module each enumeration value set must be given a unique name. The same enumeration element name can be reused, possibly with different value bindings, across multiple enumeration value sets.

ICL27

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain an enumerations using the Enum specification with a duplicate Enum item.

ICL28

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain an un-terminated string constant.

ICL29

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the iApplyEndState value, defined within an alias declaration, does not contain unknown bits.

ICL30

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks for inverted string constants.

Within a constant value expression the inversion operator (~) can only be applied to numeric constant values.

ICL31

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the constant value concatenation used to specify the value of a parameter definition in a module is comprised entirely of constant value elements that are of compatible type, either all string values or all sized numeric values.

ICL32

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a parameter definition, within a module declaration, does not contain a concatenation of values with an unsized numeric constant value.

ICL33

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no self referential parameter definitions.

A parameter definition must be made in terms of explicit immediate constant values or references to other parameter definitions. When the parameter definition hierarchy is resolved it cannot contain cyclical parameter references.

ICL34

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the size preamble for a sized number is a non-zero unsigned decimal number, where the size value is a deferred constant value or a parameter reference whose value is not verified until ICL51 is activated.

ICL35

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that only module parameters (not local parameters) are overridden in a module instance.

ICL36

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that any parameter present in an instance matches the definition in the instantiated module so that the parameter can be substituted wherever it is found in the module.

Additionally, the parameter reference included in a parameter override element must match both the name and type of a parameter definition within the module being instantiated.

ICL37

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Checks that a module definition contains at least one port definition. A module without any ports is not valid within ICL.

However, Tessent Shell does not require ICL modules with InSystemTest (IST) functionality to have any port definitions. These modules are not IEEE1687-2014 compliant and might be rejected by third-party tools.

ICL38

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies several conditions related to ports in the design.

Specifically, this rule verifies the following:

- There can be at most one WriteEnPort per module.
- There can be at most one ReadEnPort per module.

ICL39

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all range specifications of an ICL port have the same index direction.

Within a module, the same base name can be used when defining different vector port elements. However, all of the vector port definitions sharing the same base name must use a consistent index direction (either ascending or descending).

ICL40

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module has at most one definition of a port using a scalar only name. If multiple bits are required, then they must be vector bits.

ICL41

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the range specifications of an ICL port do not overlap.

Within a module, the same base name can be used when defining different vector port elements. However, all of the vector port definitions sharing the same base name must have mutually disjoint index ranges.

ICL42

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that elements referenced within a ScanInterface are ports.

ICL43

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the module that declares a scan input port for a client interface also declares a shift enable control port or a TMS port for that client scan interface. A client scan interface is any scan interface that doesn't contain host scan interface control ports.

ICL44

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a scan-in port is present in the module if the module declares either a scan register or a scan out port.

ICL45

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a module with a ToIRSelectPort also has a TMSPort.

ICL48

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies the input port bindings on an instance within a module. Each bit of each input port binding must correspond to an input port bit definition on the instantiated module. Where an input port binding has a vector range constraint the underlying module port must have a port definition with a compatible definition.

ICL49

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all module input ports that require input port bindings on an instance of a module have the bindings specified.

Table 7-14 lists the port types that require input binding.

Table 7-14. Port Types and Required Input Binding

Port Type
DataInPort ¹
AddressPort ²
WriteEnPort ³
ReadEnPort ⁴
ScanInPort

1. When the instance does not have the AddressValue property.
2. When the instance does not have the AddressValue property.
3. When the instance does not have the AddressValue property.
4. When the instance does not have the AddressValue property.

ICL50

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that each bit of the module instance input port bindings within a module is specified only once. It is not valid to specify more than one input source binding for any bit of a module input port in a module instance.

ICL51

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the width of a sized constant value is greater than 0 (zero).

This check is similar to [ICL34](#), however here the rule also considers the actual resolved values of any parameter references.

ICL52

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the size of a constant value expression or a reference to a named element within the module is compatible with the size expected at the location that the expression or reference is used.

For example a data register source value must be no wider than the width of the data register itself. This check takes into account unsized expression concatenation filling and truncation behaviors when considering the size compatibility.

ICL53

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the name referenced from a parameter reference corresponds to the name of a parameter or a local parameter within the same module.

ICL54

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the type of a parameter being de-referenced matches the type expected by the context of the point of de-reference. For example, within a range constraint of a vector signal “foo[\$p]” the type of the parameter p must be an unsigned integer.

ICL55

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the parameter reference included in a parameter override element matches both the name and type of a parameter definition within the module being instantiated.

ICL56

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that any instance output port reference matches an output port on an instance within the module.

ICL57

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that any reference to an element as an rvalue makes reference to a name that exists within the same module, or an output or a module instantiated within that module. It is not permitted to make reference to hierarchical aliases. Where the reference uses a vector range constraint, it must actually reference an object that is itself a vector entity with a compatible range. References to module local names must resolve correctly to a readable entity.

ICL58

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that for a known size target, only one unsized number is included with a group of sized numbers. For an unknown size target, no unsized number can be included in concatenation.

ICL59

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a differential inverse clock is only an inverse of a primary clock. Additionally, the clock port name, an input port connection of a clock port name is required and must be a concat_clock_signal with a matching width and, if differential, a corresponding differential source.

ICL60

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all instances are instances of modules for which a module definition has been read. It is not valid to use an ICL module hierarchy that contains references to modules that have not been defined.

ICL61

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that attribute values are either strings or sized binary values or unsigned integers.

ICL62

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that when the iApplyEndState option is present, each bit of the resolved alias is a suitable target for an iWrite, specifically a DataInPort or a register bit.

ICL63

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies for a resolved alias with iApplyEndState value that includes register bits that each such bit has a reset value, and that the reset value matches the iApplyEndState value.

ICL64

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that connected ResetPorts and ToResetPorts have consistent ActivePolarity specifications.

ICL65

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the source of an rvalue is of a suitable type to drive the location that the rvalue is being consumed.

ICL66

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that no component has itself as a member of its fanin.

ICL67

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a ToIRSelectPort only connects to the select input of a 2:1 ScanMux.

ICL68

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that within each module_def that all scan signals have a path from at least one scanInPort_name and to at least one scanOutPort_name.

ICL70

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

`report_drc_rules`: Supported

Verifies that a scan interface is composed of the correct subset of signals along with control signals related by direction.

Specifically:

- Scan Client Interface — {ScanInPort, ScanOutPort, SelectPort, CaptureEnPort, ShiftEnPort, UpdateEnPort}
- Scan Host Interface — {ScanInPort, ScanOutPort, ToSelectPort, ToCaptureEnPort, ToShiftEnPort, ToUpdateEnPort}
- TAP Client Interface — {ScanInPort, ScanOutPort, TMSPort, TRSTPort}
- TAP Host Interface — {ScanInPort, ScanOutPort, ToTMSPort, ToTRSTPort}

Scan client, scan host, TAP client and TAP host ports may not be mixed.

ICL71

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

`report_drc_rules`: Supported

Verifies that the anonymous ScanInterface can be correctly assembled in an ICL module with a ScanInPort and/or a ScanOutPort but without explicit ScanInterface specifications.

This check applies only to the ICL module associated with the current design.

If an ICL module has a ScanInPort and/or a ScanOutPort but does not have an explicit ScanInterface specification, then Tessent Shell creates the anonymous ScanInterface from the existing ScanInPort, ScanOutPort and the ScanInterface control ports of the ICL module. ICL71 checks if this ScanInterface creation is possible as well as unique.

Each ICL71 message starts with the following lines:

```
// Error: ICL module 'test' contains ports of type 'ScanInPort' and ports
// of type 'ScanOutPort', but no ScanInterface specification. However,
// the combination of specified scan ports and ScanInterface control
// ports does not allow an unambiguous assembling of the anonymous
// ScanInterface.
// The module has the following problematic ports:
```

followed by a list of affected ports and explanations of what is wrong (called “port_list” in the following).

Each ICL71 message ends with the source location of the affected module:

```
// Found on or near line 1 of file '../data/icl/test1.icl'. (ICL71-1)
```

The anonymous ScanInterface cannot have more than one ScanInPort or more than one ScanOutPort. The existence of more than one ScanInPort or more than one ScanOutPort in a module without explicit ScanInterface specification makes the anonymous ScanInterface ambiguous. In this case, port_list consists of the following:

```
// 3 ScanInPorts (si1, si2, si3), explicit ScanInterface specification
// required.
// 2 ScanOutPorts (so1, so2), explicit ScanInterface specification
// required.
```

The next step looks for ScanInterface control ports indicating ScanInterfaces of conflicting types. Each ScanInterface can be one of the following:

- client ScanInterface (which has one port with function ShiftEnPort and/or one or many ports with function SelectPort)
- host ScanInterface (which has one port with function ToShiftEnPort and/or one or many ports with function ToSelectPort)
- client-TAP ScanInterface (which has one port with function TMSPort)
- host-TAP ScanInterface (which has one port with function ToTMSPort)

Ports with function SelectPort can also be present in an ICL module with a ScanInterface other than “client”, and ports with function ToSelectPort can also be present in an ICL module with a ScanInterface other than “host”. Therefore, SelectPorts and ToSelectPorts are not part of the first step of the analysis of conflicting ScanInterface types. Only ports of type ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort are considered during this analysis. If there is more than one port function out of ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort present in the ICL module, the port_list shows the related ports:

```
// 1 ShiftEnPort (se), indicating the ScanInterface type 'client',
// 1 TMSPort (tms), indicating the ScanInterface type 'client-TAP'.
```

If there is no port with function ShiftEnPort, ToShiftEnPort, TMSPort, ToTMSPort, then the ScanInterface type must be derived from the SelectPorts and ToSelectPorts. In this situation, ICL71 checks also conflicts between ports with function SelectPort and ports with function ToSelectPort. If there is more than one port function out of SelectPort and ToSelectPort present in the ICL module, the port_list shows the following:

```
// 1 SelectPort (sel), indicating the ScanInterface type 'client',
// 1 ToSelectPort (tsel), indicating the ScanInterface type 'host'.
```

If there is no port with function ShiftEnPort, ToShiftEnPort, TMSPort, ToTMSPort, SelectPort or ToSelectPort, such that the ScanInterface creation is invalid whatever ScanInterface type is chosen, port_list contains the following:

```
// none of the mandatory ScanInterface control ports.
```

The absence of a ScanInPort or a ScanOutPort also reduces the possible choices for the ScanInterface type. Client and client-TAP ScanInterfaces must have a ScanInPort as well as a ScanOutPort, while host and host-TAP ScanInterfaces need only one of them (but may have both). If there is no ScanInPort or no ScanOutPort, but the existing ScanInterface control ports only allow ScanInterfaces of type “client” or “client-TAP” instead of “host” or “host-TAP”, then the port_list contains the following hints:

```
// one ScanInPort, no ScanOutPort. The anonymous ScanInterface must be of
// type 'host' or 'host-TAP', none of the mandatory control ports for
// ScanInterfaces of type 'host' or 'host-TAP', 1 TMSPort (tms),
// indicating the ScanInterface type 'client-TAP'.
```

If there is exactly one port function out of ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort present in the ICL module, then ICL71 still needs to check that there is exactly one port with this function. Otherwise the ScanInterface would be ambiguous. If there is more than one port of type ShiftEnPort, port_list consists of the following:

```
// 2 ShiftEnPorts (sel1, se2), the tool cannot build an anonymous client
// ScanInterface without a unique ShiftEnPort or a unique SelectPort.
```

In case of more than one port of type ToShiftEnPort:

```
// 2 ToShiftEnPorts (tse1, tse2), the tool cannot build an anonymous host
// ScanInterface without a unique ToShiftEnPort or a unique ToSelectPort.
```

In case of more than one port of type TMSPort:

```
// 2 TMSPorts (tms1, tms2), the tool cannot build an anonymous client-TAP
// ScanInterface without a unique TMSPort.
```

In case of more than one port of type ToTMSPort:

```
// 2 ToTMSPorts (ttms1, ttms2), the tool cannot build an anonymous
// host-TAP ScanInterface without a unique ToTMSPort.
```

If there are no ports with function ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort, then the ICL module must contain at least ports with function SelectPort or ToSelectPort in order to allow the creation of a valid anonymous ScanInterface. Anonymous ScanInterfaces with more than one SelectPort or more than one ToSelectPort are not created, although they would be allowed according to the standard. If there are several ports with function SelectPort, port_list is as follows:

```
// 2 SelectPorts (sel1, sel2), the tool cannot build an anonymous client
// ScanInterface without a unique ShiftEnPort or a unique SelectPort.
```

In case of more than one port of type ToSelectPort:

```
// 2 ToSelectPorts (tsel1, tsel2), the tool cannot build an anonymous
// host ScanInterface without a unique ToShiftEnPort or a unique
// ToSelectPort.
```

The summary message for ICL71:

```
ICL71: #fails=1 handling=error (assembling of anonymous ScanInterface failed)
```

ICL75

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the instance output referenced from the Port specification of a one hot scan group or one hot data group is a port with an enable control or that is a port that is driven by a one hot scan group or one hot data group, respectively.

The net effect of this check is to ensure that any port reference will be a reference to an element that has a tri-state driver, either directly (because the port is tri-state enabled) or indirectly because it references a port that is driven by a one hot group.

ICL76

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about ResetValues for data storage elements.

Specifically:

- Data storage elements controlling scan mux must have ResetValue.
- Data storage elements enabling ScanOutPort must have ResetValue.

ICL77

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that a ScanMux or OneHotScanGroup does not deselect a controlling ScanRegister.

Reconfigurable Scan Networks may run into deadlocks if scan registers, which control the scan path configuration, take themselves out of the active path. In such a situation, the scan path through the register cannot be activated without altering the values in the register, and the values in the register cannot be altered without activating the scan path through the register. In order to avoid such situations, ICL77 checks if there is a register which controls a ScanMux or a OneHotScanGroup and which is only accessible through a subset of the inputs of the ScanMux/OneHotScanGroup.

However, such a configuration can be intended, therefore the severity of that rule can be lowered to “Warning”. If a scan path deadlock has happened, it can usually be resolved by a Global Reset or a Local Reset, unless the ResetValue specifications are chosen such that they establish a deadlock from the very beginning.

The ICL77 check does not deal with scan paths which would never be selected by the retargeter anyway, such as scan paths that are unconnected or tied off. It still may be possible to configure the scan network explicitly (by means of iWrite and iApply), such that a deadlock happens, for example, by selecting such an unconnected or tied scan path. ICL77 mainly aims at the prevention of accidental deadlocks which the retargeter could run into.

ICL78

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a ScanRegister that drives either the data input of a ScanMux whose select input comes from a ToIRSelectPort must be in the active scan chain in either one or the other state of the ToIRSelect signal, but not both.

The two inputs of the ToIRSelect scan mux are considered up until the point at which they reconverge. Scan registers in one path cannot be in the other path.

ICL79

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

When the iApplyEndState option is present, then multiple iApplyEndState values must not specify incompatible values for any target. If there are multiple iApplyEndState values for a single target bit then they must all be of the same value.

The ultimate source of the iApplyEndState value must come directly from a data register, scan register or primary input, there can be no other intervening logic in that path.

ICL80

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that no component has itself as a member of its fan-in.

This rule is similar to ICL66, but it also considers circular paths across hierarchy boundaries, while ICL66 only considers circular paths within one ICL module.

ICL81

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The eventual driving source of a port of type {ReadEnablePort, WriteEnablePort or AddressPort} can only be a primary input or a scan register.

ICL82

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The value associated with a clock frequency adjustment, either multiplication or division, must be a positive integer (greater than 0).

ICL83

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The names associated with a hierarchical alias definition must all be resolvable (they must exist) to entities of an appropriate type and refer to things that have definitions compatible to their usage within the hierarchical alias definition. For example if the reference in the alias definition includes a range constraint then the resolved entity must be a vector object with compatible range definition.

ICL84

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no unknown bits in a constant value expression and that the evaluation of the expression does not cause an integer overflow or a division by zero.

ICL85

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a LogicSignal expression containing an operand with unknown bits.

ICL86

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a LogicSignal expression containing operands with incompatible widths. For example, 1'b0 | 2'b11.

ICL87

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks for problems with enumeration references. For example, an enum value is not defined or a register has no enum references but uses it.

ICL88

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no circular instance hierarchies in the ICL design.

If a circular loop is found, the error message also indicates the location of the loop.

```
// Error: Loop in module instantiation tree:  
//         Module 'm1' has instance 'i2' of module 'm2'.  
//         Module 'm3' has instance 'i1' of module 'm1'.  
//         Module 'm2' has instance 'i3' of module 'm3'.  
//         ICL hierarchies with circular instance loops are not supported.  
Found on or near line 12 of file '../data/icl/test.icl'.  
// ICL88
```

This is a module level hierarchical check that is performed only once the set_current_design command has been issued.

ICL89

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

ICL connectivity extraction can be directed by the value of certain special attributes associated with the ICL ports for which a connection is being extracted, such as the ‘connection_rule_option’ attribute.

For more information about directing ICL extraction with ICL attributes see “[Attributes of the ICL Extraction Flow](#)” semantic in the *Tessent IJTAG User’s Manual*. This semantic rule checks that the ICL port connection rule attribute has a value that is compatible with the port direction and will fail if this is not the case.

An error message is issued if the attribute ‘connection_rule_option’ has a value of ‘allowed_no_destination’ for an input port, or one of the following values for an output port: ‘allowed_no_source’, ‘allowed_tied’, ‘allowed_tied_low’, ‘allowed_tied_high’.

For example:

```
// Error: ICL module 'block1' declares the port 'broken_i1' with
attribute 'connection_rule_option' that has the invalid resolved value
'allowed_no_destination'. For an input port the value must be one of:
'allowed_no_source', 'allowed_tied', 'allowed_tied_high' or
'allowed_tied_low'.
// Error: ICL module 'block1' declares the port 'broken_o1' with
attribute 'connection_rule_option' that has the invalid resolved value
'allowed_no_source'. For an output port only the value
'allowed_no_destination' is permitted.
// Error: ICL module 'block1' declares the port 'broken_o2' with
attribute 'connection_rule_option' that has the invalid resolved value
'allowed_tied'. For an output port only the value
'allowed_no_destination' is permitted.
// Error: ICL module 'block1' declares the port 'broken_o3' with
attribute 'connection_rule_option' that has the invalid resolved value
'allowed_tied_low'. For an output port only the value
'allowed_no_destination' is permitted.
// Error: ICL module 'block1' declares the port 'broken_o4' with
attribute 'connection_rule_option' that has the invalid resolved value
'allowed_tied_high'. For an output port only the value
'allowed_no_destination' is permitted.
```

ICL90

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a problem with the tessent_scan_register attribute.

A scan register with the tessent_scan_register attribute is not allowed to specify a capture source, reset value or default load. If a scan register with the tessent_scan_register attribute specifies a range constraint, it must be in descending order with a right index of 0. A scan register with the tessent_scan_register attribute cannot be the driver of a data out port or a logic signal.

ICL91

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a problem with the tessent_signature attribute.

If the module contains a tessent_instrument_type attribute which defines the module to be an instrument of type Mentor, the tessent_signature attribute must be present and must contain a valid signature.

ICL92

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link name declarations. Within an ICL module each access link must be given a unique name.

ICL93

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link instruction declarations. Within an ICL module each access link instruction must be unique.

ICL94

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains only one specified BSDL entity in the access link. Within an ICL module a BSDL entity can be specified only once in an access link.

ICL95

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode scan path declarations. Within an ICL module an opcode scan path can be specified only once in an access link.

ICL96

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode active signal set declarations. Within an ICL module an opcode active signal set can be specified only once in an access link.

ICL97

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no access link declarations if the module does not specify the top module. Only the top ICL module can have access link declarations.

ICL98

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains access link declarations that are associated with valid paths.

ICL99

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that mux selection conditions are not made redundant by previous selections in the same mux.

ICL100

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate chain names within a scan interface.

ICL101

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a ScanInterface contains ports that allow its unique classification into one of the following four ScanInterface types: client, host, client-TAP or host-TAP.

ICL102

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about addressable data registers.

Specifically:

- Modules containing readable addressable data registers must have a ReadEnPort.
- Modules containing writable addressable data registers must have a WriteEnPort.

ICL103

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that each ScanInPort and each ScanOutPort of the top module is referenced explicitly in at least one ScanInterface specification, if the module contains more than one ScanInPort and/or more than one ScanOutPort.

This check applies only to the ICL module associated with the current design.

ICL104

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Each ‘scan client’ ScanInterface statement in a blackbox module must include a valid DefaultLoadValue definition, which reflects the length of the scan chain behind each ScanInterface after a global reset.

ICL105

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the ports in a ScanInterface conform to certain rules.

Specifically:

A Scan Client Interface shall have

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function ShiftEnPort and/or one or many ports with function SelectPort
- zero or one port with function CaptureEnPort, without function_modifier attribute
- zero or one port with function UpdateEnPort, without function_modifier attribute
- optional port(s) with function CaptureEnPort and distinct function_modifier attributes
- optional port(s) with function UpdateEnPort and distinct function_modifier attributes

A Scan Host Interface shall have

- one port with function ScanInPort and/or one port with function ScanOutPort
- one port with function ToShiftEnPort and/or one or many ports with function ToSelectPort
- zero or one port with function ToCaptureEnPort, without function_modifier attribute
- zero or one port with functionToUpdateEnPort, without function_modifier attribute

- optional port(s) with function ToCaptureEnPort and distinct function_modifier attributes
- optional port(s) with function ToUpdateEnPort and distinct function_modifier attributes

A TAP Client Interface shall have

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function TMSPort
- (optional port with function TRSTPort)

A TAP Host Interface shall have

- one port with function ScanInPort and/or one port with function ScanOutPort
- one port with function ToTMSPort
- (optional port with function ToTRSTPort)

Note

 ScanInterfaces with multiple CaptureEnPorts, UpdateEnPorts, ToCaptureEnPorts or ToUpdateEnPorts are not standard-compliant. Third-party IJTAG tools might reject the processing of ICL files with such ScanInterface specifications.

ICL106

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a chain definition of an ICL module scan interface only contains ports of the type ScanInPort and ScanOutPorts

ICL107

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that only scan client and TAP client interfaces can have scan chain definitions. Scan host and TAP host interfaces cannot have scan chain definitions.

ICL108

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that if a client scan interface has more than one ScanInPort/ScanOutPort pair, then each pair shall be declared in a scan chain definition.

ICL109

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that if a scan chain definition is present, there are no ScanInPort, ScanOutPort or DefaultLoadValue statements outside the scan chain definition.

ICL110

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that for every scan chain definition of an ICL module's scan interface, a ScanInPort/ScanOutPort pair is defined. The ports of this chain cannot be used in any other scan chain definition in the same scan interface.

ICL111

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about data storage elements.

Specifically:

- Data storage elements controlling SelectPorts of ScanInterfaces must have ResetValue.
- Data storage elements controlling ToSelectPorts of ScanInterfaces must have ResetValue.

ICL112

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that no binary digits of value 1 are truncated from a sized constant number.

If the number of binary digits of a number's binary representation is larger than the number of binary digits defined by the size constant, then the leftmost digits are truncated until it matches the size constant. Truncating a digit of value 1 generates an error.

ICL113

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies the mutual exclusivity of certain items in a ToClockPort specification.

The use of "Period" is mutually exclusive with the usage of "DifferentialInvOf", "Source", "FreqMultiplier" and "FreqDivider".

The use of "DifferentialInvOf" is mutually exclusive with the usage of "Period", "Source", "FreqMultiplier" and "FreqDivider".

ICL114

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that ports of type AddressPort, ReadEnPort and WriteEnPort are not explicitly connected in an instantiation with an AddressValue specification.

ICL115

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

This check verifies that the appropriate restrictions are fulfilled when an ICL DataMux is under certain circumstances.

If an ICL DataMux is used for the gating of an UpdateEn signal, a CaptureEn signal, a Reset signal, a TMS signal or a TRST signal, the following restrictions apply to the types of the other multiplexer inputs:

If the intercepted signal is of type captureEn signal or updateEn signal, all the other multiplexer inputs must be constant zero values.

If the intercepted signal is of type reset signal, TRST signal or TMS signal, all the other multiplexer inputs must be signals of type data signal.

See also ICL129 for more details on the use of the DataMux construct for the interception of reset signals, TRST signals or TMS signals.

ICL116

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the function _ modifier attribute is used consistently, if an ICL DataMux is used for the gating of an UpdateEn signal, a CaptureEn signal, a Reset signal, a TMS signal or a TRST signal.

ICL117

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there is no DataRegister involved in the scan path configuration. There must not be any DataRegister in the input cone of the select input of a ScanMux or in the input cone of the EnableSource of a ScanOutPort.

ICL118

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there is no scan path from the ScanInPort of a Host ScanInterface to the ScanOutPort.

A host ScanInterface with both ScanInPort and ScanOutPort provides scan data through its ScanOutPort to the instrument(s) under its control, and receives scan data through its ScanInPort from the instrument(s) under its control. Therefore there cannot be a scan path between the ScanInPort and the ScanOutPort of a host ScanInterface inside the module containing the ScanInterface.

ICL119

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the instruments which are under control of an ICL host ScanInterface do not bypass the ScanInPort or the ScanOutPort of that ScanInterface.

If a host ScanInterface has both a ScanInPort as well as a ScanOutPort, then the instrument(s) under its control must also use both of them. It is not allowed to connect the instrument(s) to the ScanInPort only and let the ScanOutPort be unused, nor is it allowed to connect the instrument(s) to the ScanOutPort only and let the ScanInPort be unused.

ICL120

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the function_modifier attribute definition of an ICL port is compatible with the function_modifier attribute definition of the source of that ICL port.

ICL121

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the value of the attribute “function_modifier” of an ICL port is one of the following: CaptureShiftClock, CaptureShiftClockInv, tap_fsm_state, tap_fsm_state_delayed.

ICL122

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the DefaultLoadValue specification and the ResetValue specification of a register match at those bits which are non-X in both values.

ICL123

Category: ICL Semantics

Rule Check Type: ICL Primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the source of certain output ports can be unambiguously implied, if it has not been explicitly specified.

If there is a ToTRSTPort without “Source” property, the module must not have more than one input port of type TRSTPort.

If there is a ToTMSPort without “Source” property, the module must not have more than one input port of type TMSPort.

If there is a ToResetPort without “Source” property, the module must not have more than one input port of type ResetPort.

ICL124

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the source of certain instance input ports can be unambiguously implied, if the instance input port binding has not been explicitly specified.

If there is an instance port of type TRSTPort without input port binding, the module that contains the instance must not have more than one input port of type TRSTPort.

If there an instance port of type TMSPort without input port binding, the module that contains the instance must not have more than one input port of type TMSPort.

If there is an instance port of type ResetPort without input port binding, the module that contains the instance must not have more than one input port of type ResetPort.

ICL125

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the module contains the required input ports for the implied sources of certain module output ports.

If there is a ToTMSPort without “Source” property, then the module must also have a TMSPort. The ToTMSPort will then be assumed to be connected to that unique TMSPort. ICL123 ensures there is only one TMSPort in that case.

ICL126

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the module contains the required input ports for the implied sources of certain instance input port bindings.

If there is an instance without explicit input port binding for a port of type TMSPort, then the parent module must also have a TMSPort. The instance input port of type TMSPort will then be assumed to be connected to that unique TMSPort of the parent module. ICL124 ensures there is only one TMSPort in that case.

ICL127

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Verifies that a module containing registers with ResetValue property has a unique reset signal.

In order to allow the unambiguous association between registers and their local reset source, there must be at most one reset signal in a module containing DataRegisters or ScanRegisters with ResetValue property. The ICL127 check considers module input ports of type ResetPort and instance output ports of type ToResetPort as a “reset signal”.

If the module does not have a reset signal, then the registers with ResetValue property only react to the Global Reset.

ICL128

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that an ICL module with addressable instances and/or addressable data registers also has AddressPort specifications.

Currently, explicit input port bindings of the AddressPorts of an addressable instance are not supported (see ICL114). An addressable instance can only be instantiated in a module with AddressPort specifications, and those specifications are used to provide the address value to the addressable instance. The ultimate source of the address signal (scan register or primary input) must be declared outside of the module that contains the addressable instance.

ICL129

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the multiplexers, which are part of the control logic of reset signals, TRST signals or TMS signals, select the appropriate input after a reset is performed.

The ICL construct DataMux can be used to intercept reset signals, TRST signals and TMS signals. This is required, for example, for the implementation of a Local Reset or for the instantiation of child TAPs. Such a “specialized” DataMux has exactly one data input with a special function (reset, TRST, TMS), while the other inputs are connected to data signals providing, for example, the local reset value. The ultimate sources of the select input(s) of the DataMux must be either primary inputs or registers with ResetValue. If the ResetValues of the registers are chosen such that the DataMux does not select the input with the special function (reset, TRST, TMS), then the selected data signal must evaluate to the active state of the intercepted signal (any of 0 or 1 in case of a TMS signal) after a Global Reset.

This rule ensures that the control logic of Reset signals, TRST signals and TMS signals is always in a defined state, and it ensures that the Global Reset (the event triggered by the iReset command) always affects all parts of the ICL network. When the ICL network complies with this rule, the Local Reset circuitry cannot accidentally block the Global Reset.

ICL130

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that there is no reference to an output port from within the ICL module.

Unlike Verilog, the IEEE 1687-2014 standard does not allow the specification of an output port as the source of a signal within the ICL module. Tesson IJTAG can be instructed to tolerate this configuration by setting the handling of this DRC to “warning”, “note” or “ignore”.

ICL131

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that each scan register bit serves at most one of the purposes defined below.

These are:

- Scan network reconfiguration
- Local Reset or isolation of embedded TAPs
- Application of the iOverrideScanInterface –capture and iOverrideScanInterface –update specifications for internal scan interfaces (not implemented yet)

ICL132

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

Verifies that vector port specifications with the same base name have consistent port directions.

ICL allows the specification of different port functions for disjoint subsets of the port’s index range. However, all port specifications with the same base name must have either input port functions or output port functions.

ICL133

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

Verifies that vector port specifications with the same base name form a continuous index range.

If the specification of a vector port is split into several parts, the union of the specified index ranges must be a continuous range without gaps. The DRC handling of this rule can be set to error in order to enforce strict standard compliance of the ICL module specification.

ICL134

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

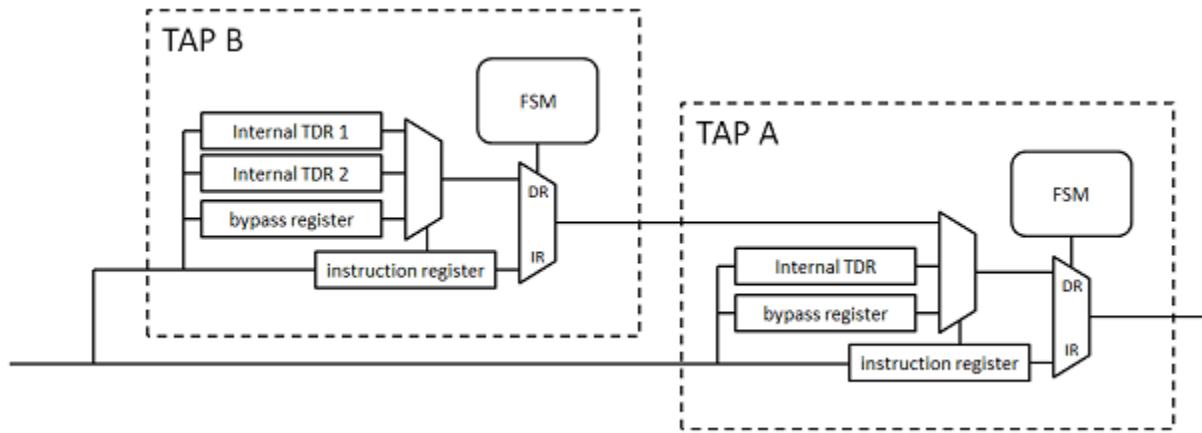
Default Handling: Error

report_drc_rules: Supported

Verifies the accessibility of IR/DR multiplexers in both types of scan loads, IR scan loads as well as DR scan loads.

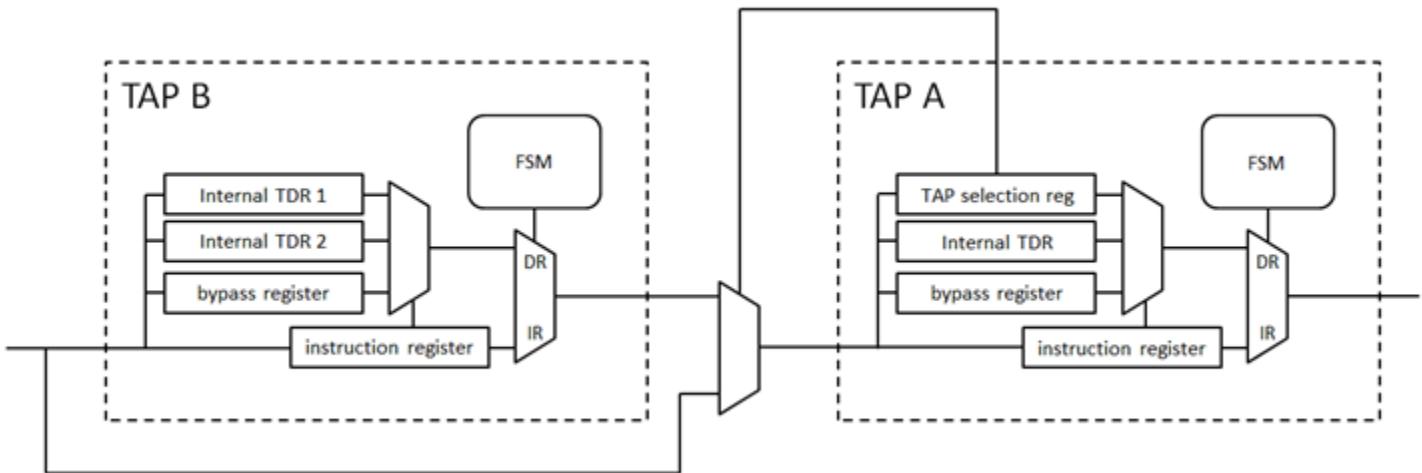
In a design with multiple TAP controllers, the active TAP controllers are controlled simultaneously. All state machines of the active TAP controllers are operated such that they are all in the same TAP state. Consequently, it is not possible to perform, for example, a scan load through the instruction register of an embedded TAP, while the master TAP is in Shift-DR state. This scenario is shown in [Figure 7-46](#). The instruction register of TAP controller “TAP B” is not accessible, because the configuration of the scan path for the access of the register requires the TAP controller “TAP A” to be in a DR state. However, the two TAP controllers cannot be in different states at the same time.

Figure 7-46. Invalid multi-TAP Configuration



To prevent such inaccessible registers, ICL134 verifies that IR/DR multiplexers are not on a pure DR-path or on a pure IR-path of another (superordinate) TAP controller. Each IR/DR multiplexer must be accessible in DR scan loads as well as in IR scan loads. [Figure 7-47](#) shows a valid multi-TAP configuration.

Figure 7-47. Valid multi-TAP Configuration



ICL135

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies the consistency between the type of a scan interface and the type of the data that is shifted through the scan interface.

There are four types of scan interfaces, which can be identified by the port functions of their controlling ports:

- Client ScanInterface (CaptureEnPort, ShiftEnPort, UpdateEnPort, SelectPort, ResetPort)
- Host ScanInterface (ToCaptureEnPort, ToShiftEnPort, ToUpdateEnPort, ToSelectPort, ToResetPort)
- Client-TAP ScanInterface (TMSPort, TRSTPort)
- Host-TAP ScanInterface (ToTMSPort, ToTRSTPort)

Based on the top level scan interfaces and the IR/DR multiplexers of the TAP controllers, ICL135 verifies the following:

- Each scan interface, which is only accessible during DR scan loads, must be an ordinary (that is, a non-TAP) scan interface. All ordinary (non-TAP) scan interfaces must only be accessible during DR scan loads. (A scan load through an ordinary top level scan interface is automatically considered a DR scan load).
- Each scan interface, which is accessible during DR scan loads as well as during IR scan loads, must be a client-TAP scan interface or a Host-TAP scan interface. All client-TAP scan interfaces and all host-TAP scan interfaces must be accessible during IR scan loads as well as during DR scan loads.
- There are no scan interfaces for the exclusive use with IR scan loads.

ICL136

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

For backward compatibility reasons, Tessent IJTAG accepts some ICL constructs like LaunchEdge, which are not part of the IEEE 1687-2014 standard, but which have been part of a previous draft version. This check fails if such a deprecated ICL construct is encountered. The DRC handling can be set to “error” in order to ensure strict standard compliance.

ICL137

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

This DRC rules out certain scenarios in which different ScanInterfaces of the same ICL module cannot be activated independently of each other. This check considers only the presence of SelectPorts in client ScanInterface specifications and the presence of ToSelectPorts in host ScanInterface specifications, it does not check if the SelectPorts can actually be controlled in such way that either the one or the other ScanInterface is active.

If two client ScanInterfaces contain SelectPorts, then the set of SelectPorts in one of the ScanInterfaces must not be equal to and must not be a subset of the set of SelectPorts in the other ScanInterface. The same holds for host ScanInterfaces and ToSelectPorts.

EDT Rules (K Rules)

EDT rules only apply to designs that incorporate EDT technology. They are generally performed by Tessent TestKompress when switching from Setup system mode. The tool also performs some checks at other times such as in response to a write_edt_files command or before writing EDT files. These rule checks find EDT design definition inconsistencies.

The following subsections describe each of the EDT rules and the special handling you can set for them.

K1	2896
K2	2896
K3	2896
K4	2897
K5	2897
K6	2897
K7	2899
K8	2899
K9	2899
K10	2901
K11	2901
K12	2902
K13	2902
K14	2902
K15	2903
K16	2908
K17	2908
K18	2909
K19	2909
K20	2910
K21	2911
K22	2911
K23	2913
K24	2913
K25	2914

K1

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT requires a circuit with scan chains. This rule check verifies that the design contains scan chains. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K2

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT does not support designs with more than one scan group. This rule check verifies that the design defines exactly one scan group. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K3

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This rule check verifies that the scan chain input and scan chain output pin of each scan chain are both connected to either top-level ports or to internal pins.

The following are valid ways to connect the scan chains:

- In the IP Creation Phase, all of the scan chains must be connected to primary inputs (PIs) and primary outputs (POs).
- In the Pattern Generation Phase, the scan-in and scan out pin of each chain must be connected either to a PI and PO (for an uncompressed scan chain) or to internal nodes (for a compressed scan chain driven by and observed through the EDT logic).

The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K4

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

The EDT decompressor provides test patterns through unidirectional outputs to the inputs of the internal scan chains. Similarly, the EDT compactor accepts test responses through unidirectional inputs from the scan chain outputs. This rule check verifies that bidirectional top-level scan chain pins do not exist. The default handling for this rule violation is error.

K5

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies the existence of all required EDT control and channel pins at the top level of the design. The default handling for this rule violation is error.

If K5 reports as nonexistent an EDT control or channel pin you believe exists, it is typically because the pin does not exist at the *top* level of the design from which the tool can control the pin, or because the pin's current name differs from the pin name the tool expected. This difference could occur, for example, if you manually edited the pin name within the netlist after the IP Creation Phase. Use the [set_edt_pins](#) command to provide the tool with the correct pin name in these cases.

Note

 You can use the [set_edt_pins](#){edt_pin} - option to explicitly tell the tool that an EDT signal is driven internally and does not have an associated top-level pin. Specifying this option prevents the tool from generating a K5 violation for that pin.

K6

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

EDT pins, when shared with a design's functional pins, must be shared with a functional pin of the same type. For example, when an EDT input pin is shared with a functional pin, the functional pin must be an input. Similarly, an EDT output pin must only be shared with a

functional output pin. This rule check verifies that each EDT pin that is shared with a functional pin is of a type consistent with the functional pin. Sharing with bidirectional pins is not allowed, and is covered by other design rule checks.

The K6 rule check also verifies the EDT control and channel pins have valid names. To be consistent with Tessent FastScan and to provide immediate feedback if you change the default pin names, Tessent TestKompress checks the validity of the pin names and pin directions when you enter a [set_edt_pins](#) command. Because the default EDT pin names might also conflict with names of existing pins, the tool performs all checks again when you switch from setup mode to analysis mode. The default handling for this rule violation is error.

The tool performs K6 rule checks at three different times. The items checked at each time by this rule are as follows:

These six requirements are checked immediately after the tool reads an EDT pin name. An error message results if the stated requirement is violated:

1. The pin name must not be hierarchical.
2. If the pin name is shared with a core pin, the directions must match.
3. If the pin is part of a bus, the base name for the pin must not be the same as the name of a core pin.
4. The pin name must not be the same as the base name of a core bus.
5. If the pin name matches a core bus pin name, the index must be within the same range.
6. If a pin name is a bus bit, the bus must exist in the core. Presently, bus-based EDT pins are not supported.

In addition to the preceding six requirements, the K6 rule checks the following requirements during DRC:

1. Each EDT pin must not be shared with either another EDT pin or with a restricted core pin. For information about restricted pins, refer to “[EDT Control and Channel Pins](#)” in the *Tessent TestKompress User’s Manual*.
2. Two EDT pins must not share the same name.
3. An EDT output channel pin must not be shared with a bidi.

Finally, when you write out the files that implement the EDT IP using the [write_edt_files](#) command, the K6 rule results in a warning message for either of the following conditions:

1. The EDT clock pin is shared with a core bus pin.
2. An EDT pin is shared with a core pin that is empty (no fan-ins or fanouts).

K7

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

EDT does not support either bidirectional or tristate scan channel output pins. This is because Tessent TestKompress includes a multiplexer between the EDT IP and the output pad when a scan channel output pin is shared with a functional output pin. This rule check verifies that neither bidirectional nor tristate scan channel output pins exist. The default handling for this rule violation is error.

K8

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

This rule check verifies that neither bidirectional scan channel input pins nor bidirectional EDT control pins exist. This is necessary because the EDT hardware has no control over the signal direction of bidirectional pins, and they might be driven by the core logic during load_unload, resulting in contention problems. The default handling for this rule violation is warning.

K9

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Top-level scan chain pins cannot be shared with a design's functional pins. This is because Tessent TestKompress connects the scan chain pins to the EDT IP and they are no longer available as primary inputs or outputs in the new top level. This rule check verifies that scan chain pins are not used as functional pins. The default handling for this rule violation is warning.

The rule check assumes pin sharing in the following situations:

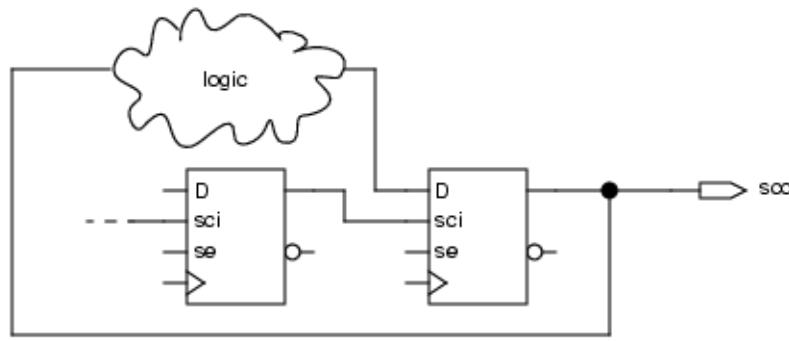
- Scan inputs — If a scan input drives a primary output, drives more than one scan cell, or is a bidirectional pin, pin sharing is assumed.
- Scan outputs — If the last scan cell of a scan chain does not drive other scan cells, and also drives exactly one primary output, pin sharing is assumed.

Note

The K9 rule check will not catch all cases of inappropriate pin sharing (the following discussion describes why this is the case). Therefore, be sure during scan insertion that you insert dedicated scan pins. This is true even if the output of the last scan cell in a scan chain is connected directly to a primary output.

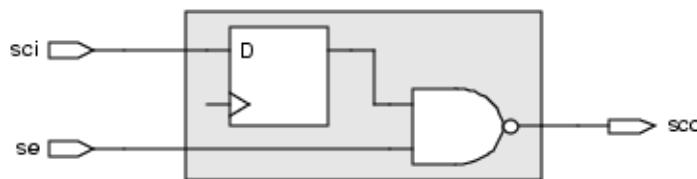
In some cases, a gate-level netlist does not contain the information to enable Tessent TestKompress to determine if there is pin sharing. For example, [Figure 7-48](#) shows a design where the last cell of a scan chain drives functional logic as well as the scan output pin. It is impossible to extract the information from the gate-level netlist about whether the pin was there before scan insertion or has been introduced for scan purposes.

Figure 7-48. Failing identification

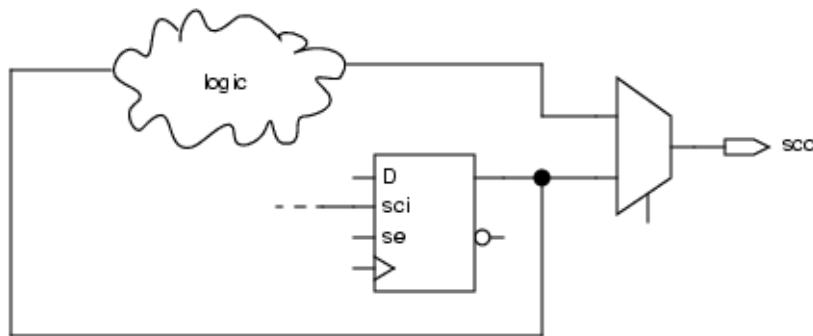


[Figure 7-49](#) shows another case where the K9 rule check cannot determine if pin sharing occurs. The scan cell contains a logic gate that drives the scan output. By examining the netlist, it cannot be determined if the logic gate is part of the scan cell or part of functional logic. Therefore, the design rule check would not detect the sharing.

Figure 7-49. Common cell structure



For the same reason, the design rule check would fail also for the circuit shown in [Figure 7-50](#).

Figure 7-50. Failing identification

K10

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Top-level scan chain pins should not be part of a functional bus. A DRC violation is reported if top-level scan chain pins are part of a functional bus. This DRC does not apply to internal scan pins.

Note

 Top-level scan chain pins can be part of a pure scan bus.

K11

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

The EDT clock, update, and bypass pins must not be shared with any other clock or RAM control signal.

The reasons for this restriction are as follows:

- If the EDT clock or EDT update are shared with a scan clock, the scan cells would be disturbed during the load_unload procedure.
- If the EDT clock, update or bypass are shared with RAM control signals, RAM sequential patterns and multiple load patterns may not be applicable.

- If the EDT clock or EDT bypass are shared with a non-scan clock, the test coverage might decrease because the EDT clock is constrained to its off state during the capture cycle.

This rule check verifies that the EDT clock, update, and bypass pins are not shared with any other clock or RAM control signal.

K12

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

To guarantee that the EDT IP is not disturbed during capture, the EDT clock must be constrained to its off-state during the capture cycle. This rule check verifies that the EDT clock pin does not drive any logic.

Note

 If the EDT clock is shared with a functional pin, the pin constraint may result in lower test coverage.

K13

Category: EDT

Contexts Supported: dft -edt

Default Handling: Note

report_drc_rules: Supported

This rule check reports all pins that will be added to the EDT top-level wrapper in order to implement the EDT hardware.

Note

 The tool reports scan channel and control pins as new pins even if a scan chain pin with the same name exists in the core circuit; otherwise that scan chain pin would not have been connected to the wrapper.

K14

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In order to prevent EDT from assigning values to the scan chain inputs during the capture cycle, all internal scan chain inputs are automatically constrained to X by EDT. This rule check verifies that there are no user-defined constraints on the scan chain inputs that are different from TIE-X.

K15

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

set_drc_handling: Not supported

This DRC verifies several EDT circuit characteristics.

Specifically:

1. Verifies that all scan channels have the proper top-level pins. All scan channels, except for blocks set up for input channel broadcasting or input channel sharing, require a dedicated top-level pin.
2. Verifies that all EDT control pin sharing is set up properly.
3. Verifies that the internal scan chains driven by the EDT decompressor are not sharing scan inputs.

The tool reports error messages slightly differently depending on whether the EDT block is inside a core. If an EDT block is not inside a core, the tool uses a format that only includes the block name as shown in the following example error messages.

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//           Block "blk1" EDT clock
//           Block "blk1" EDT update
```

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//           Block "blk1" EDT clock
//           Block "blk2" EDT update
```

- Verifies that the top-level input port does not connect to the control signal and channel input of the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//      Block "blk1" EDT update
//      Block "blk1" EDT input channel 1
```

- Verifies that the top-level input port does not connect to the control signal and channel input of different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//      Block "blk1" EDT update
//      Block "blk2" EDT input channel 1
```

- Verifies that the top-level input port does not connect to different channel inputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//      Block "blk1" EDT input channel 1
//      Block "blk1" EDT input channel 2
```

- Verifies that the top-level output port does not connect to different channel outputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//      Block "blk1" EDT output channel 1
//      Block "blk1" EDT output channel 2
```

- Verifies that the top-level output port does not connect to channel outputs on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//      Block "blk1" EDT channel output 1
//      Block "blk2" EDT channel output 1
```

- Verifies that a top-level input port does not broadcast to both compressed and uncompressed blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "scan_in1" for EDT channel and
//      uncompressed scan chain: (K15-1)
//      Block "blk1" EDT input channel 1
//      Scan chain "chain1" input
```

- Verifies that two scan chains driven by the EDT decompressor do not share the same scan input pin. If this is not true, the following error message is reported:

```
// Error: Incorrect sharing of pin "/scan_input1": (K15-1)
//           Scan chain "chain1" input
//           Scan chain "chain2" input
```

To support channel sharing between identical and non-identical blocks, this DRC does the following:

- Verifies that the control channels on non-identical blocks do not share the same top-level port. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//           Block "blk1" EDT input channel 1
//           Block "blk2" EDT input channel 1
//           When broadcasting channels to non-identical blocks, the
//           shared channels at core level cannot include any EDT
//           control registers.
```

- Verifies that if two non-identical EDT blocks have the same input channel, they also have the same “pulse_edt_before_shift_clocks” signal. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//           inconsistent pulse_edt_before_shift_clocks options:
//           (K15-1)
//           Block "blk1" has pulse_edt_before_shift_clocks set to "on"
//           Block "blk2" has pulse_edt_before_shift_clocks set to
//           "off"
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same number of external pipeline stages. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//           different number of input pipeline stages: (K15-1)
//           Block "blk1" EDT input channel 1 has 4 pipeline stages
//           Block "blk2" EDT input channel 1 has 5 pipeline stages
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks of
//           shared input channels with different input channel
//           inversions: (K15-1)
//           Block "blk1" EDT input channel 1 has no inversion
//           Block "blk2" EDT input channel 1 has inversion
```

- If an EDT block is inside a core, the tool includes the parent core name of the EDT block in the error message. This example verifies that if two input channels on two non-identical blocks (either from the top-level, from the same core, or from different cores) share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in2" for blocks of
// shared input channels with different input channel
// inversions: (K15-1)
// Block "blk1" EDT input channel 2 has no inversion.
// Block "top_core1_unwrapped_blk1" (from core instance
// "top_core1") EDT input channel 2 has inversion.
// Block "top_core2_unwrapped_blk1" (from core instance
// "top_core2") EDT input channel 2 has inversion.
// Block "top_core2_unwrapped_blk2" (from core instance
// "top_core2") EDT input channel 2 has inversion.
```

Identical Block Checks

When checking two blocks that are identical, the K15 DRC separates the conditions into two sets:

- Conditions related to the EDT IP, which include the following:
 - The number of scan chains in each block.
 - The number of input/output channels.
 - The decompressor structure and the connections between decompressor and scan chains.
 - The compactor structure and the connections between scan chains and compactors.
 - The low power controller.
- Conditions related to the embedded logic, which includes everything outside EDT IP, such as scan chain lengths and external pipeline stages.
 - Scan chain lengths.
 - Input /output pipeline stages.
 - Input channel inversions.
 - Input channel port ordering.

If the tool encounters a K15 DRC error sharing channels among non-identical blocks, the tool first checks if the set of conditions related to EDT IP is same or not for two blocks. If they are not the same, then the tool issues the error messages noted above.

If, however, the EDT IPs are identical, then the embedding logic must be different. The intention of users was possibly to use the channel broadcasting rather than channel sharing. In this case, the tool reports the following K15 error messages:

- Two corresponding chains do not have the same lengths.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because // they are
// embedded differently in the design. One of the differences is:
// Chain #1 ("chain1") of block "B1" has 5 scan cells while chain #1
// ("chain6") of block "B2" has 4 scan cells.
```

- Two corresponding EDT input channels have different pipeline stages.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared. Broadcasting all input channels
// including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block "B1" has 2 pipeline stages while input
// channel #1 of block "B2" has 3 pipeline stages.
```

- Two corresponding EDT output channels have different pipeline stages.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Output channel #1 of block "B1" has 2 pipeline stages while
// output channel #1 of block "B2" has 3 pipeline stages.
```

- Two corresponding EDT input channels have different inversions.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.// In this case,
// the two EDT controllers are identical, but they are not
// considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block "B1" has inversion while input channel
// #1 of block "B2" has no inversion.
```

- Two blocks share the same set of input channel ports but not in the same order.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block "B1" is driven by port "A" while input
// channel #1 of block "B2" is driven by port "B".
```

K16

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies that the EDT clock signal has been added as a clock using the add_clocks command.

K17

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies the EDT clock is constrained to its inactive state. This is required in order to avoid disturbing the EDT IP during the capture cycle.

K18

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule check verifies three required properties of the test procedure file.

Specifically:

1. Analyzes the off states of the EDT control signals.

In the IP Creation Phase, Tessent TestKompress creates the EDT IP such that all the control signals are active high. This check verifies that all control signals are, in fact, active high.

Note

 If a control signal (EDT clock, for example) is inverted between the chip pin and the EDT IP, and you defined the inversion using the “[set_edt_pins](#)” on page 2095 command in the IP Creation Phase, the signal at the chip level may be active-low.

2. Analyzes the EDT update control signal in the load_unload and shift procedures.

In order to operate correctly, the EDT update signal is asserted during load_unload before the leading edge of the EDT clock signal. EDT update must be reset after the leading edge of EDT clock in load_unload and before the leading edge of EDT clock in the first shift procedure. It does not matter if EDT update is reset before or after the trailing edge of EDT clock in load_unload.

3. Analyzes the EDT clock signal in the shift procedure.

This check verifies that the EDT clock is pulsed in the shift procedure in order to decompress vectors provided at the scan channel inputs.

K19

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check simulates the EDT decompressor netlist by applying pseudorandom patterns and compares the results with the emulated values. If a simulation-emulation mismatch occurs, the DRC performs diagnostic checks to narrow down the number of possible problem sources.

This design rule check also detects missing channel input constraints if a pipeline stage is clocked by a non-EDT clock.

How to Debug K19 Violations

For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, it is typically sufficient to use the `set_gate_report` command to specify Drc_pattern reporting:

`set_gate_report drc_pattern procedure_name`

Then use the `report_gates` command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K19 rule check as reference points in its automated analysis of K19 mismatches) are reported as part of the K19 error message and provide clues to where the problem may lie.

When you need to see the specific simulation values applied in each shift cycle, you can view the simulated values for all K19-simulated gates, not just the monitor gates by setting the gate report to K19:

`set_gate_report drc_pattern k19`

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

 Use “`set_gate_report drc_pattern k19`” reporting only when necessary. K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.

For additional in-depth information on how to reduce the occurrence of K19 violations, refer to the “[K19 through K22 DRC Violations](#)” and “[Understanding K19 Rule Violations](#)” sections of the *Tessent TestKompress User’s Manual*.

K20

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check identifies the number of pipeline stages within the compactors, based on simulation. It also considers channel output pipelines; reporting any discrepancy between the number of identified and specified pipeline stages between the scan chains and pins (including compactor and channel output pipelines).

Note

-  If the K19 rule check that verifies the operation of the decompressor fails, then the K20 rule check will not be able to identify the number of pipeline stages.
-

If this rule check fails, you can still use diagnostic information from the K22 design rule check to help identify a problem. Design rule K22 is checked whether the K20 rule check fails or not.

Note

-  Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.
-

K21

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In the IP Creation Phase, this design rule check verifies that lockup cells will be synthesized if there is a scan chain whose first scan cell captures data at, or after, the rising edge of the EDT clock. The rule check also verifies that lockup cells will be synthesized if the output of the last scan cell changes at, or before, the rising edge of the EDT clock. If bypass logic is included, the rule verifies that lockup cells will be synthesized between the last scan cell of one chain and the first scan cell of another chain when they are connected by the bypass logic, and the output of the last scan cell changes before, or at the same time, the first scan cell captures.

In the Pattern Generation Phase, this rule check verifies the first scan cell of each chain does not capture scan data at the same time the corresponding EDT decompressor output changes its signal. If the EDT compactor is pipelined, the rule check also verifies the compactor does not capture at the same time the output of the last scan cell in each chain changes. This could result in clock skew problems, as the EDT and the core circuitry are part of different clock domains. If a violation is reported, you should add lockup cells or modify the timing. The default handling for this rule violation is warning. The K21 rule check will not be able to identify timing problems in the Pattern Generation Phase if the K19 rule check, which verifies the operation of the decompressor, fails.

K22

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check simulates the EDT compactor fed by pseudorandom patterns, and compares the results to the emulated values.

Two different checks are performed, as follows:

1. The tool applies a pseudorandom pattern to the compactor inputs, with all scan chains enabled.
2. The tool tests each compactor input separately. In this case, one pseudo-random pattern is applied to the selected compactor input, and the compactor is programmed to mask all other inputs (scan chains). Decoder problems and scan chain permutations are detected by this test.

If a simulation-emulation mismatch occurs, the tool automatically performs diagnostic checks to eliminate possible sources of the mismatch. Messages regarding K22 violations usually incorporate relevant information from these diagnostic checks to help you find the source of the mismatch.

Note

 Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.

How to Debug K22 Violations

For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, it is typically sufficient to use the **set_gate_report** command and specify Drc_pattern reporting:

set_gate_report drc_pattern procedure_name

Then use the **report_gates** command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K22 rule check as reference points in its automated analysis of K22 mismatches) are reported as part of the K22 error message and provide clues to where the problem may lie.

Note

 In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see ““[How to Report Gate Data](#)” on page 2648.

When debugging incorrect compactor mask settings for specific patterns simulated by the design rule check, you can view the simulated values for any K22-simulated gate, not just the monitor gates, by setting the gate report to K22:

set_gate_report drc_pattern k22

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

-  Use “set_gate_report drc_pattern k22 reporting” only when necessary. K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.
-

For additional in-depth information on how to reduce the occurrence of K22 violations, refer to the “[K19 through K22 DRC Violations](#)” and “[Understanding K22 Rule Violations](#)” sections of the *Tessent TestKompress User’s Manual*.

K23

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

This design rule check verifies (when the tool generates level-sensitive EDT hardware) that there is no scan chain whose first or last scan cell contains edge-triggered flip flops.

Tessent TestKompress can generate EDT IP based on either edge-triggered flip flops or level-sensitive latches. By default, it generates EDT hardware based on flip flops, with a single clock. The edge-triggered EDT hardware can operate with a design that has scan chains based on edge-triggered scan cells or level-sensitive scan cells. Depending on the clocks used by the first and last scan cells in each chain, the tool generates lockup cells when necessary to prevent clock skew problems.

For pure LSSD designs, the tool can optionally generate EDT hardware based on level-sensitive latches with a master clock and a slave clock. But the level-sensitive EDT hardware requires a design that uses only level-sensitive scan cells, with an appropriate clock scheme for the master and slave clocks. Such a scheme ensures there is no clock skew between the EDT hardware and the first and last scan cells, so the tool does not add any lockup cells.

Note

-  If you use level-sensitive EDT hardware with bypass logic, you must make sure that scan chain outputs connected to scan chain inputs via bypass logic do not require lockup cells.
-

K24

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note, Warning, and Error

report_drc_rules: Not Supported

This design rule check is performed when EDT Finder is enabled, which it is by default, or when you have enabled the set_edt_mapping command and are reusing block-level dofles for top-level pattern creation. When EDT mapping is enabled, this rule check verifies that the clock, read control, write control, and pin constraint signals specified in the block-level dofles are consistent with the signals specified at the top level.

Note

 K24 does not run when core mapping is used, regardless of whether EDT Finder is on or off.

When core mapping is used, the connections are automatically extracted to the top and connection mapping issues are reported by other DRCs.

The tool performs this verification by tracing connectivity between specified block-level signals (ports) and primary input or output pins. The tool verifies clock off states and pin constraint values while compensating for inversions in the path. During this rule check, all EDT signals that exist as ports on the block, but not as top-level pins, are updated to refer to the top-level input or output pins as determined by the tracing algorithm.

The K24 rule check outputs three different types of messages depending on the severity level of the results:

- **Note** — K24 outputs an informational note in cases such as when an EDT channel pin name is provided and a pin of that name exists at the top-level but not at the block-level or, in the case where the number of traced pipeline stages along the channel I/O do not match user-specified values.
- **Warning** — K24 issues a warning in cases such as when the traced top-level signal is different from the user-specified signal.
- **Error** — K24 issues an error message in cases such as when an I/O pin or clock port cannot be found.

K24 verifies the existence of the specified channels and maps block-level channels to top-level ports (unless EDT Finder has been disabled). K24 severity is set to Error (unless EDT Finder has been disabled) and you cannot override this severity setting.

For detailed information about creating a top-level dofle for the modular Tessent TestKompress flow, see the [Modular Compressed ATPG](#) section of the *Tessent TestKompress User's Manual*.

K25

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

All block-level EDT pins must be functionally connected in the netlist or have a connection specified during the integration session.

K25 verifies that each block-level EDT pin is either functionally connected in the netlist or has a connection specified. If no connection exists or is specified, an error is returned.

The error message is:

```
// Error: Update pin in block "B1" is not driven in the netlist and does  
not have a defined connection. (K25-2)
```

Procedure Rules (P Rules)

The test procedure file for each scan chain group is checked to ensure adherence to the format rules and accuracy of the test procedure data. All violations of procedure rules are treated as error conditions and cannot be changed—with the exception of rules P30, P31, P32, and P33, which can be ignored.

P1.....	2918
P2.....	2919
P3.....	2919
P4.....	2920
P5.....	2920
P6.....	2920
P7.....	2921
P8.....	2921
P9.....	2921
P10.....	2922
P11.....	2922
P12.....	2923
P13.....	2924
P14.....	2924
P15.....	2925
P16.....	2925
P17.....	2925
P18.....	2926
P19.....	2926
P20.....	2927
P21.....	2927
P22.....	2927
P23.....	2928
P24.....	2928
P25.....	2929
P26.....	2929
P27.....	2929
P28.....	2930
P29.....	2931
P30.....	2931

P31.....	2932
P32.....	2932
P33.....	2932
P34.....	2933
P35.....	2933
P36.....	2934
P37.....	2934
P38.....	2934
P39.....	2935
P40.....	2935
P41.....	2935
P42.....	2936
P43.....	2936
P44.....	2937
P45.....	2937
P46.....	2938
P47.....	2938
P48.....	2938
P49.....	2939
P50.....	2939
P51.....	2939
P52.....	2940
P53.....	2940
P54.....	2941
P55.....	2942
P56.....	2942
P58.....	2943
P59.....	2943
P60.....	2943
P62.....	2944
P63.....	2944
P64.....	2945
P65.....	2945
P66.....	2945
P70.....	2946

P71.....	2946
P72.....	2947
P73.....	2947
P74.....	2947
P75.....	2948
P76.....	2949
P77.....	2949
P78.....	2950
P79.....	2950
P80.....	2951
P81.....	2952
P82.....	2952
P83.....	2953
P84.....	2953
P85.....	2954
P86.....	2954
P87.....	2955
P88.....	2955
P89.....	2956
P90.....	2956
P91.....	2957
P92.....	2957
P93.....	2958
P94.....	2958

P1

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Each statement in the test procedure file must have the proper syntax. A syntax error occurs for a statement if there is an incorrect number of arguments or an incorrect ending character (“=” for procedure statements and “;” for all other statements). You can correct this error condition by editing the indicated line of the test procedure file.

The error message is:

```
Error: syntax error near token (Token). (P1)
```

Where Token represents the token in the input file close to where the error occurred.

P2

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

For statements inside a procedure, the time of the statement must not be less than the time of a preceding statement. You can correct this error condition by editing the time value on the indicated line of the test procedure file.

The error message is:

```
Time value T less than preceding time value. (P2)
```

Where T is the time defined in the specified statement.

P3

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The P3 rule checks if there is any event in the load_unload procedure at the boundary of the apply shift procedure. You can correct this error by editing the time value on the indicated line of the test procedure file.

The error message is:

```
// The following occurred at line L:  
//Error: time value not greater than preceding time value for apply  
procedure. (P3-1)
```

L is the line number in which the failure occurred.

P4

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

All procedures must end with an **end** statement. To correct this error condition, add an **end** statement at the indicated line of the test procedure file.

The error message is:

```
Line number L, P procedure not ended. (P4-1)
```

L is the line number in which the failure occurred, and P is the procedure name.

P5

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Procedure definitions follow the syntax, procedure proc_type [proc_name] = For most procedures, the procedure name is optional and it is ignored. Procedures of type seq_transparent, clock, and sub_procedure require a name. This error occurs if one of these procedures is missing its name. Also, this error occurs if the user tries to define more than 32 clock or seq_transparent procedures.

```
Incorrect or missing name for procedure of type P. (P5)
```

Where P represents the procedure type that the error occurred on. Unknown procedure types do not cause a P1 violation.

P6

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may define a procedure only once in a single test procedure file. You can correct this error condition by deleting the duplicated procedure at the indicated line of the test procedure file.

The error message is:

Duplicate procedure P. Original procedure from file F. (P6)

Where F is the file that the first occurrence of the procedure P was found in.

P7

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Statements (except the **procedure** statement) can only execute when a procedure definition is still active. To correct this error condition, add a **procedure** statement prior to the indicated line of the test procedure file.

The error message is:

Line number L, no active procedure for S statement. (P7-1)

L is the line number in which the failure occurred, and S is the type of statement.

P8

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **load_unload** procedure must contain an **apply shift** statement. To correct this error condition, add an **apply shift** statement at the appropriate place in the **load_unload** procedure of the test procedure file.

The error message is:

Line number L, no apply shift in load_unload procedure. (P8-1)

L is the line number of the end of the load_unload procedure.

P9

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **shift** procedure must contain a **force_sci** statement. To correct this error condition, add a **force_sci** statement at the appropriate place in the **shift** procedure of the test procedure file.

The error message is:

```
Line number L, no force_sci in shift procedure. (P9-1)
```

L is the line number of the end of the shift procedure.

P10

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **shift** procedure must contain a **measure_sco** statement. To correct this error condition, add a **measure_sco** statement at the appropriate place in the **shift** procedure of the test procedure file.

The error message is:

```
Line number L, no measure_sco in shift procedure. (P10-1)
```

L is the line number of the end of the shift procedure.

P11

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you define a period for a procedure, then the period time must not be less than the time of the last procedure event. To correct this error condition, increase the period time for the indicated procedure.

The error message is:

```
Period T for P is less than time of last statement. (P11)
```

T represents the period, and P specifies the name of the procedure or the procedure type if it has no name.

How to Debug P11 Violations

Your test procedure has the following content:

```
set time scale 1.000000 ps ;  
  
timeplate gen_tp1 =  
    force_pi 0 ;  
    measure_po 250 ;  
    pulse clk1 500 750 ;  
    period 1000;  
end;
```

In this case, the tool reports a following message:

```
// The following occurred at line 7 in file test.testproc:  
// Error: Period 1000 for gen_tp1 is less than time of last statement.  
(P11-1)
```

In the timeplate gen_tp1, the clock pulse starts at 500 ps with a width of 750 ps, making the minimum period for the cycle 1250 ps. The period is defined as 1000 ps which ends before the clock pulse returns to its off state at 1250 ps, resulting in the P11 violation.

To resolve the violation, you must either increase the period or decrease the pulse.

P12

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The pin name argument for a **force** or **pulse** statement must be a valid pin name of a primary input or clock signal.

The occurrence message for a **force** statement is:

```
The following occurred at T, incorrect pin name P. (P12-1)
```

Where T represents the token in the input file near to where the error occurred and P is the pin name.

The occurrence message for a **pulse** statement is:

```
Incorrect pin [or path] name, P. (P12-1)
P is not a valid clock pin.
```

Where P is the pin or path name.

How to Debug P12 Violations

Edit the pin or path name on the indicated line of the test procedure file.

If the pin or path name refers to a clock signal, the clock must be defined using the `add_clocks` command.

P13

Category: Procedure

Contexts Supported: `dft -scan`, `patterns -scan`, `patterns -scan_diagnosis`

Default Handling: Error (cannot be changed with the `set_drc_handling` command)

`report_drc_rules`: Not Supported

For the **force** statement, you may only use the force values 0, 1, X, and Z. To correct this error condition, edit the force value on the indicated line of the test procedure file.

The error message is:

```
The following occurred at T, incorrect force value V. (P13-1)
```

Where T represents the token in the input file near to where the error occurred and V is the incorrect value.

P14

Category: Procedure

Contexts Supported: `dft -scan`, `patterns -scan`, `patterns -scan_diagnosis`

Default Handling: Error (cannot be changed with the `set_drc_handling` command)

`report_drc_rules`: Not Supported

You may only use the **force_sci** statement in the **shift** procedure. To correct this error condition, delete the **force_sci** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, force_sci only allowed in the shift procedure. (P14-1)
```

L is the line number in which the failure occurred.

P15

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **force_sci** statement once in the **shift** procedure. To correct this error condition, delete the **force_sci** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate force_sci statement. (P15-1)
```

L is the line number in which the failure occurred.

P16

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **measure_sco** statement in the **shift** procedure. To correct this error condition, delete the **measure_sco** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, measure_sco only allowed in the shift procedure. (P16-1)
```

L is the line number in which the failure occurred.

P17

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **measure_sco** statement once in the **shift** procedure. To correct this error condition, delete the **measure_sco** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate measure_sco statement. (P17-1)
```

L is the line number in which the failure occurred.

P18

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **apply shift** statement in the **load_unload** procedure. To correct this error condition, delete the **apply** statement on the indicated line of the test procedure file. Note that this DRC does not check apply statements used for subprocedures.

The error message is:

```
Line number L, apply only allowed in load_unload procedure. (P18-1)
```

L is the line number in which the failure occurred.

P19

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You cannot have two **apply shift** statements with a value greater than 1 in the **load_unload** procedure.

The error message is:

```
Line number L, duplicate apply shift statement. (P19-1)
```

L is the line number in which the failure occurred and P19 is the rule ID number.

Note

 You can repeat the **apply shift** statement for as many serial shifts as necessary; however, you must specify 1 for the value in the additional **apply shift** statements.

P20

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **apply shadow_control** statement in the **load_unload** procedure. To correct this error condition, select the **apply shadow_control** statement on the indicated line of the test procedure file.

The error message is:

Line number L, duplicate apply shadow_control statement. (P20-1)

L is the line number in which the failure occurred.

P21

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **apply shadow_control** statement may only be used immediately after the **apply shift** statement. To correct this error condition, move the **apply shadow_control** statement from its current position on the indicated line of the test procedure file to the position following the **apply shift** statement.

The error message is:

Line number L, apply shift must precede apply shadow_control. (P21-1)

L is the line number in which the failure occurred.

P22

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You must set the number of repetitions for the **apply shadow_control** statement to 1. To correct this error condition, change the repetition argument of the **apply shadow_control** statement on the indicated line of the test procedure file to the value of 1.

The error message is:

```
Line number L, repetitions for apply shadow_control must be 1. (P22-1)
```

L is the line number in which the failure occurred.

P23

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the **apply** statement for the **shift** and **shadow_control** procedures. To correct this error condition, delete the **apply** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, apply procedure P not allowed. (P23-1)
```

L is the line number in which the failure occurred and P is the procedure name.

P24

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

The timing in a port specific pulse or force statement for a clock must specify the correct number of pulse statements to match the frequency multiplier. Correct this error condition by editing the timing for the clock identified.

The error message is:

```
Specified timing for frequency multiplied clock C in timeplate T only has P pulses where R are required. (P24-1)
```

C is the clock name, T is the timeplate, P is the number of pulses and R are the required number of pulses.

P25

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

You may only use the **initialize** command at time 0 of the **test_setup** procedure. Correct this error condition by moving the statement, on the indicated line of the test procedure file, to the beginning of the **test_setup** procedure.

The error message is:

```
Line number L, initialize command can only be used at time 0  of
test_setup procedure. (P25-1)
```

L is the line number in which the failure occurred.

P26

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The instance name argument for the **initialize** statement must correspond to at least one primitive-level latch or flip-flop gate instance. Correct this error condition by editing the statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, incorrect instance name N. (P26-1)
```

L is the line number in which the failure occurred and N is the instance name.

P27

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This rule checks two aspects of force values on clock pins.

Specifically:

- The only allowed force values for a clock pin are 0 and 1. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

The error message is:

```
// Error: Line number L, clock C may not be forced to a V. (P27-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and V is the incorrect value.

- The only allowed force state for a clock pin is OFF when pulsed in the timeplate. To fix a P27 error where a clock pin with pulse timing is forced to its ON state, create another timeplate that does not pulse the clock and use this timeplate for the procedure or cycle in question.

The error message is:

```
// The following occurred at line L in file F:
```

```
// Error: Clock C which is pulsed in timeplate T may not be forced  
// to on state. (P27-1)
```

F is the file name, L is the line number in which the failure occurred, C is the clock pin name, and T is the timeplate name.

- The **force_capture_clock_on** and **force_capture_clock_off** procedure file statements are used in a procedure timeplate that contains pulse statements:

```
// The following occurred at line L in file procfile_name.
```

```
// Error: Cannot use force_capture_clock_on in procedure when  
// referenced timeplate timeplate has pulse statements. (P27-1)
```

P28

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The only allowed force value for a write control pin is its defined off-state value, unless it is also defined as a clock. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

The error message is:

```
// The following occurred at line L:  
// Error: Write control W may not be forced to a v. (P28-1)
```

L is the line number in which the failure occurred, W is the write control pin name, and V is the incorrect value.

P29

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

All clocks must be at their off-state prior to any pattern which places a clock line at an on-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file.

Note

 Partially overlapping clocks are not supported.

The error message is:

```
// The following occurred at line L:  
// Error: Clock C not at off-state prior to clock_on pattern. (P29-1)
```

L is the line number in which the failure occurred and C is the clock pin name.

P30

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not place a clock at its on-state at the same time it forces a non-clock pin to a value or place another clock at its off-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

 Partially overlapping clocks are not supported.

The error message is:

```
// The following occurred at line L:  
// Error: Clock C cannot be forced on at this time. (P30-1)
```

L is the line number in which the failure occurred, and C is the clock pin name.

P31

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not force a non-clock pin to a value at the same time it forces a clock pin to a value. Correct this error condition by changing **force** statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

The error message is:

```
// The followin offured at line L:  
// Error: Non-clock pin N cannot be forced at this time. (P31-1)
```

L is the line number in which the failure occurred, and N is the non-clock pin name.

P32

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not place a clock at its off-state at the same time it places another clock at its on-state. Correct this error condition by changing **force** statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

The error message is:

```
// Error: Line number L, clock pin C cannot be forced off at this time.  
(P32-1)
```

L is the line number on which the failure occurred, and C is the clock pin name.

P33

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When a pattern places a clock at its off-state, all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state at the same time as the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

 Partially overlapping clocks are not supported.

The error message is:

```
// The following occurred at line L:  
// Error: Clock C not at off-state at end of clock_off pattern. (P33-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and P33-1 is the rule and violation ID number.

P34

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

At the end of all test procedures (except **test_setup** procedure), all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state prior to the indicated line of the test procedure file.

The error message is:

```
// The following occurred at this line L:  
// Error: Clock C not off at end of P procedure. (P34-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and P is the procedure name.

P35

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

At the end of the **master_observe** and **shadow_observe** procedures, all constrained pins must be at their constrained states. The tools assume they are at that state prior to the procedures. Correct this error condition by forcing the indicated pin to its constrained state prior to the indicated line of the test procedure file.

The error message is:

Constrained pin L not at constrained value at end of P procedure. (P35-1)

L is the line number in which the failure occurred, and P is the procedure name.

P36

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The test procedure file must contain a **load_unload** procedure. Correct this error condition by adding a **load_unload** procedure to the test procedure file.

The error message is:

No load_unload procedure in group procedure file. (P36-1)

P37

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The test procedure file must contain a **shift** procedure. Correct this error condition by adding a **shift** procedure to the test procedure file.

The error message is:

No shift procedure in group procedure file. (P37-1)

P38

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If the test procedure file contains an **apply shadow_control** statement, the test procedure file must contain a **shadow_control** procedure. Correct this error condition by adding a **shadow_control** procedure to the test procedure file or by deleting the **apply shadow_control** statement.

The error message is:

```
No shadow_control procedure in group procedure file. (P38-1)
```

P39

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If the test procedure file contains a **shadow_control** procedure, the test procedure file must also contain an **apply shadow_control** statement. Correct this error condition by adding an **apply shadow_control** statement or by deleting the **shadow_control** procedure in the test procedure file.

The error message is:

```
Unused shadow_control procedure in group procedure file. (P39-1)
```

P40

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you turn on the skew load option, the test procedure file must contain a **skew_load** procedure. Correct this error by adding a **skew_load** procedure to the test procedure file or by turning off the skew load option with the set_skewed_load command.

The error message is:

```
Skewed_load may not be used without a skew_load procedure. (P40-1)
```

P41

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the `set_drc_handling` command)

`report_drc_rules`: Not Supported

The values of all pins at the beginning of the **shift** procedure must be the same as at the end of the **shift** procedure. Correct this error condition by changing force statements in the **shift** or **load_unload** procedures.

The error message is:

```
P initial value different from final value in shift procedure. (P41-1)
```

P is the pin name.

P42

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the `set_drc_handling` command)

`report_drc_rules`: Not Supported

If there are multiple test procedure files, there can be no more than one **test_setup** procedure. Correct this error condition by deleting the extra **test_setup** procedures.

The error message is:

```
Multiple test_setup procedure defined in test procedure files. (P42-1)
```

If there are multiple test procedure files, there can be no more than one **test_end** procedure.

Correct this error condition by deleting the extra **test_end** procedures.

The error message is:

```
Multiple test_end procedure defined in test procedure files. (P42)
```

P43

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the `set_drc_handling` command)

`report_drc_rules`: Not Supported

You must place all write and read control lines at their off-state at time 0 of the **load_unload** procedure. Correct this error condition by adding the appropriate force statements to the test procedure file or by deleting the write or read control lines.

The error message is:

```
T control N not at off-state at time 0 of load_unload procedure. (P43-1)
```

T is the type of control (write or read) and N is the name of the control line.

P44

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only place the **restore_pis** statement at the end of a **seq_transparent** procedure. Correct this error condition by either removing the statement or by placing it at the end of a valid **seq_transparent** procedure.

The error message is:

```
Line number L, restore_pis can only be used at the end of the seq_transparent procedure. (P44-1)
```

L is the test procedure file line number where the error occurred, and P44-1 is the rule and violation ID number.

P45

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only place the **condition** statement at the beginning of a **seq_transparent** procedure. Correct this error condition by either removing the statement or by placing it at the beginning of a valid **seq_transparent** procedure.

The error message is:

```
Line number L, conditions can only be used at time 0 of seq_transparent procedures. (P45-1)
```

L is the test procedure file line number where the error occurred, and P45-1 is the rule and violation ID number.

P46

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **condition** statement must identify a pin pathname that connects to the output of a scan state element. The path between the two points can only contain buffers and inverters. Correct this error condition by either removing the **condition** statement or by correcting the pin pathname

The error message is:

```
Invalid condition for nonscan cell N (G) during procedure P. (P46-1)
```

N is the name of the non-scan cell, G is the gate ID number, P is the **seq_transparent** procedure name, and P46-1 is the rule and violation ID number.

P47

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The statement mentioned is not allowed to be used in the type of procedure that is named. For example, it is not legal to have a measure statement in a load_unload procedure.

The error message is:

```
Statement S cannot be used in P procedure. (P47)
```

Where S is the statement, and P is the name of the procedure.

P48

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The set strobe_window time statement specifies a bad value for the strobe_window time.

For more information, refer to “[Set Statement](#)” in the *Tessent Shell User’s Manual*.

The error message is:

Negative or zero strobe window specified. (P48)

P49

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The set time scale statement specifies a bad time value.

For more information, refer to “[Set Statement](#)” in the *Tessent Shell User’s Manual*.

The error message is:

Negative or zero value specified for the time scale. (P49)

P50

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The pulse width value for the named pulse statement in a named procedure or timeplate is zero or negative. The error message can appear for either timeplates or procedures, and will use either the word “timeplate” or “procedure.”

The error message is:

Negative or zero pulse width for S specified in [timeplate | procedure]
P. (P50)

Where S is the pulse statement, and P is the procedure name.

P51

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named statement occurs more than once in the named timeplate. For example, forcing the same pin twice in a timeplate is not allowed. Pulsing a clock more than once in a single timeplate is also not allowed.

The error message is:

```
Duplicate S statement in timeplate T. (P51)
```

Where S is the statement that occurs more than once, and T is the timeplate.

P52

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A pulse statement was used with a pin name which is not a clock pin. This message can also appear for either timeplates or procedures.

The error message is:

```
Non-clock pin P cannot be pulsed in [ timeplate | procedure ] S. (P52)
```

Where P is the pin name, and S is the timeplate or procedure.

P53

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named timeplate does not specify a pulse statement required by one of the procedures which uses this timeplate. For example, if a ram_sequential procedure contains a pulse_read_clock statement, but the timeplate that the procedure uses does not contain a pulse statement for any read clock, this warning will be issued. You can correct this warning by adding pulse statements for all needed clocks in the timeplate definitions.

The error message is:

```
No clock pulse timing in timeplate T which matches event E in procedure P,  
two cycle clock pulses will be used. (P53)
```

Where T is the timeplate statement, E is the pulse event, and P is the procedure.

Note

 A P53 warning will not cause the Vector Interfaces outputs to create incorrect patterns, just larger patterns. During pattern output, if a timeplate does not contain a needed clock pulse statement, the clock pulse will be created by using two cycles and the force_pi time of the timeplate.

How to Debug P53 Violations for analyze_simulation_mismatches

When using the [analyze_simulation_mismatches](#) command for a pattern with a P53 violation, the tool issues an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```

The problem of the P53 error is due to an undefined clock waveform in the used timeplate.

This violation can be seen when saving patterns as in the following example:

```
// command: write_pattern testcase_failing_patt.v -verilog -serial -all
// -rep
// Warning: No clock pulse found in timeplate tp_shift which matches
// event /RESETN in procedure used in patterns, two cycle clock pulse
// will be used. (P53)
```

This example shows that the P53 error is issued when saving the Verilog test bench.

The reason for this is the /RESETN is pulsed in some patterns, but the timeplate used for the pattern “tp_shift” does not contain a pulse time for /RESETN, which causes the pattern to expand the capture cycle into two cycles. The P53 rule changes the timing of test patterns and causes an unnecessary increase in test application.

It is recommended you fix any P53 issues first when you want to debug the mismatch issue.

P54

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Some statement has referenced a name that is not defined. For example, using the “timeplate” statement in a procedure to reference a timeplate that has not yet been defined. Or, using the “force” statement on a pin name that the ATPG tool cannot find.

The error message is:

```
Undefined identifier N referenced by S. (P54)
```

Where N is the undefined name and S is the statement referring to N.

P55

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This error message indicates that an old procedure type of statement is used within a cycle-based procedure, or that a cycle-based procedure statement is used in an old style procedure. For example, it would be illegal to put a break_repeat statement in a procedure that uses timeplates and cycles; put a force statement with a time value in a cycle-based procedure; or use old procedure statements in a new procedure (for example, the capture procedure).

The error message is:

```
Illegal mix of time based and cycle based statements in procedure P. (P55)
```

Where P is the procedure name.

P56

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named timeplate or procedure is empty. If it is a procedure, it has no statements, including no timeplate references.

The error message is:

```
[ Timeplate | Procedure ] S has no statements. (P56)
```

Where S is the named timeplate or procedure.

P58

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The time specified for a break_repeat statement is too small to be useful, for example, it is equal to one or less.

The error message is:

```
Break_repeat time value T is too small in procedure P. (P58)
```

Where T is a number, and P is the name of the procedure in which T resides.

P59

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Checks for incorrect loop values on loop statements in the procedure file.

The error message is:

```
// The following occurred at line N in file <filename>:  
// Error: In <procedure_name>, loop with repetition less than 1 is not  
// allowed. (P59-1)
```

P60

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Checks if the set tck_ratio is compatible with the timeplate used in the test_setup and/or test_end procedures.

The error message is:

```
// The following occurred at line N in <filename>:  
// Error: IJTAG error for <iCall | iReset | iMerge>: Specified tck_ratio  
// of N does not agree with timing for tck clock in timeplate <name>.  
// (P60-1)
```

For certain conditions, the tool issues optional message continuations for the error. For example:

```
// A tck ratio of 1 requires pulse timing on the tck clock.  
  
// A tck ratio of 4 or greater requires force timing on the tck clock not  
// pulse timing.  
  
// A tck ratio of 4 or greater requires force timing on the tck clock to  
// occur before measure_po time.  
  
// A tck ratio of 2 requires force timing for tck to occur after the  
// measure_po time.
```

P62

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

In a definition statement, the identifier N has already been used in a previous definition. For example, if there is a pin name in the design called “GROUP1” and there is an “alias” statement which tries to define an alias called “GROUP1,” the user will receive the following P62 message, “alias name GROUP1 already in use as pin name.”

The error message is:

```
X name N already in use as Y. (P62)
```

Where X is the type of definition, and Y is the previous definition type.

P63

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Only one occurrence of a pin is allowed in an alias statement. Pin P occurs more than once in the current alias statement.

The error message is:

```
Duplicate pin P in alias statement. (P63)
```

P64

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Pin P has a direction (input, output, bidirectional) that is not compatible with statement S.

Direction of pin P is not compatible with S statement. (P64)

For example, the following occurrence message:

Direction of pin Out1 is not compatible with force statement. (P64)

Out1 is an output pin in the netlist that is not compatible with a force statement.

P65

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A duplicate statement S has been found in procedure P. The tool will ignore the first occurrence of this statement. A duplicate statement is one that occurs at the same time and on the same pin (if applicable) as a previous statement.

For example, if a statement in a procedure causes pin A to be forced to a zero at time zero, and then a subsequent statement causes pin A to be forced to a one at time zero, the user will receive a P65 message and the first force statement will be ignored.

The error message is:

Duplicate S statement in procedure P, first statement being ignored. (P65)

S is the either just the event type or the event type and pin name when needed. P is the procedure type.

P66

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The P66 rule is used to check for missing statements within a load_unload_registers procedure. For example, if no Shift block is specified, or if an event statement using the '#' character is not present for each shift_assignment passed to the load_unload_registers procedure, a P66 is issued.

The error message is:

```
Error: Procedure procedure_name is missing required statement_string statement. (P66)
```

See “[Serial Register Load and Unload for LogicBIST and ATPG](#)” for complete information.

P70

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (can be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Source clocks defined in clock control definitions cannot be constrained.

The error message is:

```
// Warning: Source clock <name> declared in clock control <name> cannot be constrained. (P70-<occurrence>)
```

P71

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The pin pathname listed in an internal clock definition must be the pin pathname of an existing clock.

This rule verifies that the pin pathname specified by the CLOCK_CONTROL keyword in the clock definition defines an existing clock. Clocks are added with the [add_clocks](#) command. Internal clocks are added with the add_clocks command.

The error message is:

```
Error: Name "p1" used in clock control is not defined as a clock. (P71)
```

Where P1 is the pin pathname of the clock.

P72

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The global control conditions and source clocks defined in multiple clock controls of the same or equivalent clocks must be exactly the same.

If the global controls defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock controls with different global controls defined for the same clock. (P72)
```

If the source clock list defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock controls with different source clocks defined for the same clock. (P72)
```

Where P1 is the pin pathname of the clock.

P73

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A FORCE statement cannot be used to force an on value on a clock pin. P73 verifies that no clock pins are forced on in clock control definitions.

The error message is:

```
Error: Cannot force clock "p1" to its on value in clock control for clock "p2". (P73)
```

Where both P1 and P2 are the pin pathnames of the clocks.

P74

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When multiple clock control definitions are used, compatible conditions must exist to turn off other clock pins while the clock being controlled is pulsed.

Also, a clock pin cannot be forced off when it is defined as a source clock.

This rule check verifies that the clock control conditions defined by multiple clocks are compatible and that the clock being forced to off is not defined as a source clock in the same clock control definition.

If incompatible clock conditions exist, the following error message is reported.

```
Error: Clock "P1" is explicitly forced off in clock control for clock "P2". However, there is no way to force "P1" off while pulsing "P2". (P74)
```

If the clock that is turned off is defined as the source clock in the clock control definition of the equivalent clock, the following error message is reported.

```
Error: Clock "P1" is forced to be off explicitly in clock control for clock "P2" while it is defined as source clock at the same time. (P74)
```

Where P1 and P2 are the pin pathnames of the clocks.

P75

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Clock control definitions cannot contain conditions that force the clock being controlled or an equivalent clock to an off state.

If a FORCE statement is used to force a controlled clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for the same clock. (P75)
```

If a FORCE statement is used to force an equivalent clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for its equivalent clock "p2". (P75)
```

Where P1 and P2 are the pin pathnames of the clocks.

P76

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

At least one of several components must be defined in a clock control definition.

Specifically:

- ATPG_CYCLE #, END block
- ATPG_SEQUENCE, END block
- ATPG_SEQUENCE <N> <M>, END block
- FORCE statement that forces the non-clock pin to off
- CONDITION statement

If none of the statements/procedures listed are present, one of the following warnings display:

Warning: The clock control for clock "P1" is unconditional for ATPG_CYCLE "p2". (P76)

Warning: The clock control for clock "P1" is unconditional for ATPG_SEQUENCE applied to all capture cycles. (P76)

Warning: The clock control for clock "P1" is unconditional for ATPG_SEQUENCE "p1" "p2". (P76)

Where P1 is the pin pathname of a clock and P2 and P3 are capture cycle numbers. The tool assumes that the clock is always pulsed in the specified capture cycles when test patterns are generated.

P77

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A non-clock pin cannot be defined as a source clock.

This rule check verifies that the pin pathname specified by the SOURCE_CLOCK keyword is a clock pin.

If the pin pathname specified by the SOURCE_CLOCK keyword is not a clock pin, the following error message is reported.

```
Error: Cannot define non-clock pin "p1" as SOURCE_CLOCK. (P77)
```

Where P1 is the pin pathname of the clock.

P78

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Multiple sequence blocks for a clock must contain local force or condition statements.

If a {ATPG_SEQUENCE, END} or {ATPG_SEQUENCE <N> <M>, END} block is declared without any local force or condition statement, the following error message is reported.

```
Error: Multiple ATPG_SEQUENCE blocks without local condition are defined  
in clock control for clock "P1". (P78)
```

Where P1 is the pin pathname of the clock.

P79

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A clock control definition containing both sequence and per-cycle blocks must provide mutually exclusive force and/or condition statements to control clock pins.

If an ATPG_SEQUENCE block is declared both with and without force or condition statements, the following error message is reported.

```
Error: ATPG_SEQUENCES with and without local condition are defined in  
clock control for clock "P1". (P79)
```

If an ATPG_SEQUENCE block is declared without force or condition statements and an ATPG_CYCLE block declares force or condition statements, the following error message is reported.

```
Error: Both ATPG_SEQUENCE mode without local condition and per-cycle mode with local condition are defined in clock control for clock "p1". (P79)
```

Where P1 is the pin pathname of the clock.

P80

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Conditions defined to pulse clocks in multiple clock control block definitions must be mutually exclusive.

This rule check verifies that:

- Sequence clock control conditions that pulse the clock are mutually exclusive with the per-cycle clock control conditions that pulse the same clock.
- Sequence clock control conditions that pulse the clock are mutually exclusive of the clock control condition used to pulse the same clock in another sequence definition.

If the clock control condition to pulse a clock in sequence block is not mutually exclusive with the clock control condition for the same clock in a per-cycle block a P80 violation is reported. Depending on the violation, one of the following error messages is reported.

```
Error: In clock control for clock "P1", the conditions defined in ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions defined in ATPG_CYCLE P4. (P80)
```

```
Error: In clock control for clock "P1", the conditions defined in ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions defined in ATPG_SEQUENCE P4 P5. (P80)
```

```
Error: In clock control for clock "P1", the conditions defined in ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions defined in ATPG_SEQUENCE applied to all capture cycles. (P80)
```

```
Error: In clock control for clock "P1", the conditions defined in ATPG_CYCLE "p2" are not mutually exclusive with the conditions defined in ATPG_SEQUENCE applied to all capture cycles. (P80)
```

Where P1 is pin pathname of the clock and P2, P3, P4, and P5 are capture cycles.

P81

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

There must be a disabled condition defined for sequence clock control definitions, either by applying a clock control condition or setting the source clock to the off state.

This rule check verifies that a clock controlled by a sequence block can be disabled.

If there is no way to disable a clock controlled by a sequence block a P81 DRC violation is reported. Depending on the violation, one of the following warning messages is reported:

Warning: No way to disable sequence mode clock control for clock "P1".
(P81)

Warning: No way to disable sequence mode clock control for clock "P1" by disabling clock control condition. (P81)

Where P1 is the pin pathname of the clock.

P82

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

There must be a way to disable clocks defined with per-cycle clock control.

If there is no way to disable a clock defined with per-cycle clock control by either applying a condition or setting the source clock to the off state, the following warning message is reported:

Warning: No way to disable cycle mode clock control for "P1". (P82)

If there is no way to disable a clock defined with per-cycle clock control using a condition, but it can be disabled by setting the source clock to the off state, the following warning message is reported:

Warning: No way to disable cycle mode clock control for clock "P1" by disabling clock control condition. (P82)

Where P1 is the pin pathname of the clock.

P83

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When using per-cycle clock control, there must be a way to disable a clock after the capture cycle.

This rule check verifies the on condition for the clock during the capture cycle is mutually exclusive with the off state condition for the same clock after the capture cycle.

If there is no source clock defined or the source clock cannot be suppressed (defined as pulse-always), one of the following warning messages is reported, and the generated test patterns do not include P1 capture cycles while the clock under control is pulsed at P1.

Warning: No way to pulse clock "P1" at capture cycle "p2" while turning off the same clock after the capture cycle "p2" in cycle mode clock control for clock "P1". (P83)
Warning: No way to turn off clock "P1" after capture cycle "p2" in cycle mode clock control for clock "P1". (P83)

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported.

Warning: No way to pulse clock "P1" at capture cycle "p2" while turning off the same clock after the capture cycle "p2" in cycle mode clock control for clock "P1" by disabling clock control condition. (P83)

Warning: No way to turn off clock "P1" after capture cycle "p2" in cycle mode clock control for clock "P1" by disabling clock control condition. (P83)

Where P1 is the pin pathname of the clock and P2 is the capture cycle.

P84

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When both sequence mode and cycle mode clock controls are defined for a clock, there must be a way to disable the internal clock for both definitions.

This rule check verifies that a clock running in both sequential and per-cycle mode can be disabled.

If a clock running in both sequence and per-cycle cannot be disabled, the following warning message is reported.

Warning: No way to disable both sequence mode and cycle mode clock control for clock "p1" simultaneously. (P84)

Where P1 is the pin pathname of the clock.

P85

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When clock controls exist for two clocks not defined as equivalent, there must be a way to turn off both clocks simultaneously during a capture cycle.

If both clocks cannot be disabled simultaneously during a capture cycle, the following warning message is reported.

Warning: Clock control is unable to turn off clock "P1" and clock "P2" simultaneously at capture cycle "P3". (P85)

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P86

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not supported

When clock controls exist for two clocks that are not defined as equivalent and one clock is pulsed during a capture cycle, there must be a way to turn off the other clock during the same capture cycle.

If there is no source clock defined or all source clocks are defined as pulse-always, the following warning message is reported:

Warning: Clock control is unable to turn on clock "P1" while turning off clock "P2" during capture cycle "P3". (P86)

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported:

Warning: Clock control is unable to turn on clock "P1" while turning off clock "P2" during capture cycle "P3" by disabling clock control condition. (P86)

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P87

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A clock control cannot be defined for a pulse-always clock.

If a clock control is defined for a pulse-always clock, the following error message is reported.

Warning: Cannot define clock control for pulse-always clock "P1". (P87)

Where P1 is the pin pathname of the clock.

P88

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A CONDITION or FORCE statement cannot assign a value to a non-scan state element.

If a CONDITION or FORCE statement assigns a value to a non-scan state element, the following error message is reported.

Error: Value is forced at non-PI and non-scan cell "P1" in clock control for clock "P2". (P88)

Where P1 is the pin pathname of the gate a value is assigned to and P2 is the name of the clock under control.

P89

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not supported

When you use clock control definitions, all unconstrained internal clocks must have an associated clock control definition.

Clock controls are defined in the test procedure file and can be enabled or disabled using the [set_clock_controls](#) command.

When no clock control is defined for an unconstrained internal clock, the tool displays following error message:

```
Error: Clock control is not defined for the internal clock "P1". (P89)
```

Where P1 is the clock pin pathname.

Note that an unconstrained internal clock does not need a clock control definition and does not cause a P89 error under any of the following circumstances:

- The internal clock is defined as pin-equivalent to other clocks and one of the equivalent clocks has clock control defined.
- The internal clock is a pulse-always or pulse-in-capture clock.
- The internal clock is an asynchronous clock (defined with “add_clocks -period”).
- The internal clock is not driving any observation points (POs or scan cells).

P90

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Internal clock controls cannot be dependent on source clocks defined by other internal clock controls.

For example, when the clock control for clock clk1 defines clk2 as source clock, the clock control for clock clk2 defines clk1 as source clock.

However, you can define the clock under control as the source clock for itself to unconditionally force itself to an off state.

When no clock control is defined for an unconstrained internal clock, the following warning message is reported.

Error: Clock controls for clocks "P1" and "P2" depend on each other through source clocks. (P90)

Where P1 and P2 are pin pathnames of clocks.

P91

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not supported

Mutually exclusive control conditions cannot exist for different ATPG_CYCLES of the same clock.

If mutually exclusive control conditions are defined in different ATPG_CYCLES for the same clock, the following error message is reported:

Warning: In clock control for clock "P1", the conditions defined in ATPG_CYCLES "P2" and "P3" are mutually exclusive.

Where P1 is the clock pin pathname and P2 and P3 are capture cycles.

P92

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Verifies that added internal PI (primary input) signals are controlled deterministically.

Specifically, verifies that every internal PI signal satisfies at least one of the following criteria:

- The internal PI has a pin constraint.
- The internal PI has an ATPG constraint.
- The internal PI is a pulse-always or pulse-in-capture clock.
- If NCP are present and enabled, the internal PI must be forced all the time in the internal mode of every enabled NCP.

If none of these criteria is true, the following error message is reported:

```
//Error: Internal primary input /int_clk (42) is not controlled. (P92-1)
```

Note, the following cases are not checked:

- An internal PI that is defined as a scan chain input pin.
- An internal PI that is blocked by pin constraints.
- An internal PI that passes P89 (an internal clock that is controlled by clock control definitions).

You can use the [add_input_constraints](#) and “[add_atpg_constraints](#)” on page 96 commands to properly constrain an internal PI to the right value.

P93

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The clock under control cannot define itself as the source clock.

If this is the case, the tool issues following error message:

```
//Error: Clock '<clock_pin_pathname>' cannot be defined as source clock  
for itself. (P93)
```

P94

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Verifies a consistent view of the capture window between ATPG settings and external capture procedure options or PLL cycles to avoid generating invalid patterns.

The tool only performs the P94 DRC when you issue the “set_external_capture_procedure -capture_procedure” or “set_external_capture_procedure -PLL_cycles” commands. When you issue either of these commands in SETUP mode, the check is performed during system mode analysis. When you issue either of them in Analysis mode, the check is performed immediately.

The default handling for P94 is Error, but it can be set to Warning, Ignore, or Note. When the handling for P94 is set to warning, the tool automatically adds TX cell constraints to the potential mismatching scan cells.

When the tool forces all pin constraints and constant cell constraints and collects all external PIs and clocks that may reach scan cells directly as candidates for the P94 DRC check, it applies the following checks on each candidate:

- An external clock is pulsed in an external capture procedure.

```
// Error: External clock '/slow_clock' (89) is pulsed in external
capture procedure ext_fast_cap_proc but during pattern generation it
may not be pulsed or may be pulsed for a different number of cycles
than specified in the external capture procedure. This can lead to
simulation mismatches on 7 scan cells, one of which is
'/jn_scell_inst0/' (10297) . (P94-1)

// command: set_drc_handling P94 warn
// Warning: External clock '/slow_clock' (89) is pulsed in external
capture procedure ext_fast_cap_proc but during pattern generation it
may not be pulsed or may be pulsed for a different number of cycles
than specified in the external capture procedure. This can lead to
simulation mismatches on 7 scan cells, one of which is
'/jn_scell_inst0/' (10297) . (P94-1)
// Added 7 TX cell constraints to scan cells due to P94 DRC
violations.
```

Note

 You can fix this violation by blocking this clock from directly reaching a scan cell, or by setting P94 handing to Warning, so that the tool will automatically add TX cell constraints to mask out the affected scan cells.

- An external PI (port) is forced to a known value in an external capture procedure, but does not have a pin constraint or has a constrained value that is inconsistent with the forced value.

```
// Error: Primary input '/SMC' (10) is forced to 1 during external
capture procedure ext_pll_4_pulses but not constrained during
pattern generation. This can lead to simulation mismatches on 1 scan
cell, which is '/sff117/' (561) . (P94-2)c
```

Note

 You can fix this violation by removing the force event from external capture procedure, by adding a consistent pin constraint in ATPG, or by setting P94 to Warning, so that the tool will automatically add a TX cell constraints to mask the affected scan cells.

- An external clock is NOT pulsed in the external capture procedure, and is NOT constrained to Off state in ATPG.

```
// Warning: External clock '/CLK1' (1) is not pulsed in external capture procedure ext_cap_proc but may be pulsed during pattern generation. This can lead to simulation mismatches on 10 scan cells, one of which is '/dff4_pulse_inter/' (369) . (P94-1)
```

Note



You can fix this violation by constraining the clock to off state in ATPG.

- An external clock is pulsed while -pll_cycles is set.

```
// Note: External clock '/RCLK2' (2) is pulsed during external capture when 'set_external_capture_options -pll_cycles' is used but during pattern generation it may not be pulsed or may be pulsed for a different number of cycles than specified by the -pll_cycles switch. This can lead to simulation mismatches on 7 scan cells, one of which is '/dff07/' (248) . (P94-1)
```

- An external clock is not pulsed while -pll_cycles is set.

```
// Warning: External clock '/PHYADRI' (1) is not pulsed during capture when 'set_external_capture_options -pll_cycles' is used but during pattern generation it may not be pulsed or may be pulsed for a different number of cycles than specified by the -pll_cycles switch. This can lead to simulation mismatches on 1 scan cell, which is '/dummy_reg1/' (612) . (P94-1)
// Added 1 TX cell constraints to scan cells due to P94 DRC violations.
```

The summary message issued by “report_drc_rule -summary” command for P94:

```
P94: #fails=0 handling=error (inconsistent view between ATPG settings and external capture options)
```

To debug P94, use the “[set_gate_report](#) drc stable_capture” command to view the values that are constant throughout the capture process for the affected capture.

Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)

Prior to performing core mapping for ATPG or for Scan Pattern Retargeting, Tessent Shell performs rule checks on the design as you switch from setup to analysis mode. This design rule checking phase reads the Tessent core description (TCD) files and core instance information. The tool applies all R rules, except for R7, to the Core mapping for ATPG and Scan Pattern Retargeting functionality.

R1	2961
R2	2962
R3	2962
R4	2963
R5	2963
R6	2964
R7	2964
R8	2965
R9	2968
R10	2969
R11	2970
R12	2971
R13	2972
R14	2972
R15	2974
R16	2977
R17	2979
R18	2979
R19	2980
R20	2980
R22	2981
R23	2982
R27	2982
R28	2983

R1

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that ports defined in the core description match pins at the boundary of core instance(s).

This rule helps detect when a core description is associated with the wrong design module.

R2

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Extracts connectivity between the core-level scan pins and top-level ports, including extraction of pipelining and inversion. Then, validates the ability to shift from top-level ports to the scan inputs on the core boundary, and from the scan output pins on the core boundary to top-level ports.

How to Debug a R2 Violation

To debug R2 violations, examine the shift simulation values to determine why the scan from the core to the top level is blocked using the following command.

set_gate_report drc shift

You can use the report_gates command and the trace capability in DFTVisualizer to analyze the blockage.

You can use the analyze_drcViolation command to debug a given R2 violation graphically.

R3

Category: Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that scan input pins of non-identical cores are not shared, unless broadcasting to multiple instances of the same core type.

- When broadcasting to multiple instances of the same core, the pipelining and inversion for each set of equivalent channels must be identical.

- When patterns are loaded from the chip level, if broadcasting to multiple core instances, they must load identical values at the same time.

Verifies that a top-level pin does not drive multiple channels of the same core.

Verifies that the connection from the top-level pin to identical cores connects to the pins of each core in the same order.

R4

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Validates core-level pin constraints against capture-cycle states at the core instance boundary.

The conditions present at the core boundary during core-level pattern generation must be satisfied for each core instance when the core is instantiated in the design. To verify that the same conditions occur, this DRC validates core-level pin constraints against capture-cycle states at the core instance boundary.

Note

 This DRC only validates pin constraints that are specified by the add_input_constraints command.

To perform this validation, the tool simulates the top-level test procedures (test_setup, load_unload, and shift) and input constraints, and compares the stable (constant) values during capture to the pin constraint values at the boundary of each core. The core-level input constraints are stored in the core description file.

You can use the analyze_drcViolation command to debug a given R4 violation graphically.

R5

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning

report_drc_rules: Supported

Validates that the stable values of sequential elements at the end of test_setup at the core level are met after simulating the top-level test_setup procedure. Those are the sequential elements where the "stable_after_setup" value was non-X at the core level.

If the sequential element is no longer present in the design in the retargeting phase due to black boxing or gray boxing of the core, the sequential element's value cannot be validated and will be omitted.

Because the tool cannot determine whether mismatches between the test_setup at the core and the top level matter, this DRC defaults to warning. For example, the state of a core's TAP instruction register may vary after test_setup at the core versus top level, but this may not matter for the core's scan operation.

You can use the analyze_drcViolation command to debug a given R5 violation graphically.

R6

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies the equivalency of the load_unload and shift procedures at the core and top levels.

This DRC validates simulated top-level load_unload and shift procedures against load_unload and shift procedures at the core level.

Pins that are forced in the test_setup procedure are maintained during the load_unload and shift procedures if they are not specifically constrained in the dofile.

You can use the “[report_procedures -core core_name](#)” and the “[report_timeplate -core core_name](#)” commands to debug an R6 violation.

Changing the Default Handling

The R6 DRC can be pessimistic because you may have forced a value, either directly or indirectly, at the core level that is not relevant to the operation of the core. In this case, if you choose to change the default handling for R6 from Error to Warning, the implications of doing that depend on whether the violation was due to over-specification (tool pessimism) or whether the flagged mismatch was needed for correct circuit operation. If the value that is no longer forced from the top is not relevant, downgrading R6 has no impact.

R7

Category: Scan Pattern Retargeting

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies immediately before generation of core-level re-targetable patterns that all primary inputs (PIs) are controlled deterministically.

If any PI (port) was excluded from automatic constraints as a result of applying the `is_excluded_from_isolation_constraints` attribute (found in the Hierarchical Design Data Model [Port](#) attribute table) to it., this DRC verifies that the PI is deterministically controlled and not left for ATPG to justify to different values in different patterns.

Specifically, the tool verifies that every PI signal satisfies at least one of the following criteria:

- The PI has a pin constraint.
- The PI has an ATPG constraint.
- If NCPs are present and enabled, the PI must be forced or pulsed in all cycles of the internal mode of every enabled NCP.

Note, the tool does not check the following PIs:

- An internal PI.
- A PI that is blocked by pins and/or cells whose values are constant during capture.
- A PI that is defined as a pulse-always clock.
- A PI that is defined as an pulse-in-capture clock.

R8

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

Verifies that the core-level load_unload and shift procedures can be correctly retargeted.

In patterns -scan context, when top-level load_unload and shift procedures are present, R8 additionally checks if core-level procedures can be correctly merged with top-level procedures.

The default setting for this rule is an Error. You can change the setting to be either Warning or Note by using the set_drc_handling command.

You can use the analyze_drcViolation command to debug the R8 violations that perform tracing.

The following checks are performed during test procedure retargeting:

- The test procedure is in the core description file. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core-level procedures are missing.
```

- Top-level clock off-states are consistent with those at the core level. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Clock pin 'pinName' is driven by top-level port 'portName'. The
// core-level clock pin has an off-state 'value', but the top-level
// clock port has an off-state 'value'.
```

- The path between the core pin and the top-level pin contains only buffers and inverters. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core Pin 'pinName' cannot be traced to the top level. Tracing
// stopped at a gate 'gateName' (gateId).
```

- The core-level clock pin that should be pulsed is instead constantly forced. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level clock pin 'pinName' (core 'coreName', mode
// 'coreMode') is now constantly forced to 'value'.
```

- The core-level pin that should be forced is constantly forced to an opposite value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level pin 'pinName' (core 'coreName', mode 'coreMode')
// is forced to 'value' in the core-level load_unload procedure,
// but it is now constantly forced to 'value'.
```

- The top-level port that is constrained during load_unload can only be forced to the constraint value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core pin 'pinName' is driven by top-level port 'portName'.
// The core-level pin is forced to 'value' in the core-level
// load_unload procedure, but the top-level port is currently
// constrained to 'value' during load_unload.
```

- The top-level port is forced to a different value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed in
// cycle 'cycleNumber' 1 of the 'procedureName' procedure.
// Top-level port 'portName' has conflicting requirements from the
// existing top-level procedure and the core-level pin(s) it drives:
//
// Top-Level Core-Level Core Pin Inverted Core Mode
// ----- -----
// 'value' 'value' 'corePin' Yes 'coreName' 'modeName'
// 'value' 'value' 'corePin' 'coreName' 'modeName'
```

- A clock pulse definition is missing in the timeplate. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' which is driving the following
// core-level pin: 'pinName' (core 'coreName', mode 'coreMode')
// has no pulse definition in timeplate 'timeplateName'.
```

- The top-level clock is forced to its on-state. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' driving the following core-level
// pin: 'pinName' (core 'coreName', mode 'coreMode') is forced to
// its on-state of 'value'.
```

- The top-level clock is pulsed a different number of times in two cores. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// Top-level clock port 'portName' has conflicting pulse
// requirements from the existing top-level procedure and the
// core-level pin(s) it drives:
//
// Clock Pulses Core Pin Core Mode
// ----- -----
// 'pulseCount' 'pinName' 'coreName' 'coreMode'
// 'pulseCount' 'pinName' 'coreName' 'coreMode'
```

- The top-level procedures have conflicting end_measure requirements:

```
// Error: load_unload procedure retargeting failed.  
// Top-level procedures have conflicting end_measure requirements  
// with core-level procedures. (R8-1)
```

- The core-level procedures have conflicting end_measure requirements:

```
// Error: load_unload procedure retargeting failed.  
// Conflicting core-level end_measure requirements. (R8-1)
```

R9

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies during core extraction that a core-level capture clock connects to a matching top-level clock.

This DRC only checks clocks on the core boundary; clocks internal to the core are not validated.

If a core-level clock is constrained off, and it is not pulsed in the core-level test_setup, load_unload, shift, or test_end procedures, capture validation is not needed because the R4 DRC validates all core-level input constraints. If the core-level clock is constrained off and is pulsed in any of these procedures, the tool performs the following checks:

- Verifies that the core-level clock connects to a top-level pin (either a port or cut point defined by the add_clocks command on an internal pin).
- Verifies that the offstates of the clock and top-level pin match (taking inversion into account).

If a core-level clock is not constrained off and is a pulse-always clock, the tool does the following:

- Verifies that the core-level clock connects to a top-level pin (either port or cut point defined by the add_clocks command on an internal pin).
- Verifies that the top-level pin is defined as a pulse-always clock.
- If a clock is synchronous (not added with “add_clocks -period”), the tool verifies that the off-states match, taking inversion into account. If a clock was added with “add_clocks -period”, the tool does not compare off-states.
- If a clock is synchronous, the tool verifies that it is not connected to a top-level asynchronous free-running clock.

If a core-level clock is not constrained off, is not a pulse-always clock, and is not the capture clock, the tool does not perform any additional R9 checks.

Note

 This DRC considers a clock to be a capture clock if the clock is not blocked, is not an internal primary input, is not constrained during capture, is pulse-always or pulse-in-capture, or if one or more NCPs exists, is pulsed at least once in the active NCPs.

If a core-level clock is not constrained off, is not a pulse-always clock, and is a capture clock, the tool performs the following additional checks:

- Verifies that a core-level clock connects to a top-level pin (either a port or cut point defined by the `add_clocks` command on an internal pin).
- Verifies that the offstates of a clock and top-level pin match (taking inversion into account).

For pattern retargeting when a core-level pin is a pulse-in-capture clock (that is, added with the `add_clocks -pulse_in_capture` command and switch), then the tool performs the following additional checks:

- If top-level clock is `always_pulse` issue, the tool issues an R9 violation.
- If top-level clock is `[async]` free-running, the tool issues an R9 violation.

See “[Scan Pattern Retargeting](#)” in the *Tessent Scan and ATPG User’s Manual*.

For core reuse when a core-level pin is a pulse-in-capture clock (that is, added with the `add_clocks -pulse_in_capture` command switch), then the tool performs the following additional checks:

- If the top-level clock has no special property (that is, `always_capture`, `async_free_running`, and so on), then the tool updates the clock, which means the tool maps and applies the core-level property to the top-level clock.
- In other cases the tool issues an R9 and makes no changes to the top-level clock.

See “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan and ATPG User’s Manual*.

R10

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that top-level ICL is available when any of the core instances being retargeted use IJTAG in their test_setup or test_end procedures. This check also applies if any EDT IP instances have been added. If you have disabled IJTAG, then this DRC is not run by the tool.

To correct this error condition, you should use the [extract_icl](#) command to extract top-level ICL.

By default, this violation is an error and can be downgraded to warning or note. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the tool bypasses any remaining IJTAG retargeting.

This DRC reports the following possible error messages:

```
// Error: IJTAG was used in the test_setup and test_end procedures of
//         core 'piccpu_maxlen16' (mode 'internal')
//         but top-level ICL was not extracted or read.
//         Use the 'extract_icl' command to extract top-level ICL or read
//         the top-level ICL model. (R10-1)
// Error: There was 1 R10 violation (Top-level ICL not extracted while
//         IJTAG used in cores).

// Note: IJTAG was used in the test_setup and test_end procedures of
//        core 'piccpu_maxlen16' (mode 'internal')
//        but top-level ICL was not extracted or read.
//        The PDL will not be automatically retargeted and appended to
//        the top-level procedure. (R10-1)
```

R11

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that every core instance that uses IJTAG at the core-level matches an ICL instance at the top-level. This includes design core instances in which IJTAG was used in test_setup and/or test_end, as well as any EDT instrument.

By default, this violation is an error and can be downgraded to warning or note for non-scan instruments. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the IJTAG retargeting capability skips any core instance with IJTAG that does not have a corresponding ICL instance at the top level. If the IJTAG instrument has a scan interface, it cannot be ignored during scan operations and the R11 will be automatically upgraded to an error during the scan identification process.

This DRC reports the following possible error messages:

```
// Error: Current design (mode 'internal', ICL module 'picccpu_maxlen16')
//         is expected to have ICL instance 'tessent_occ_i_clk'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-1)
// Error: Core 'picccpu_maxlen16' (mode 'internal', ICL module //
//         'picccpu_maxlen16')
//         is expected to have ICL instance 'picccpu_maxlen16_edt_i'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-2)
// Error: Core instance 'chip_int_1/picccpu_2' (core 'picccpu_maxlen16',
//         mode 'internal') uses IJTAG but has no ICL instance
//         corresponding to it in the ICL model.
//         Rerun ICL extraction and provide the missing ICL model. (R11-3)
// Error: There were 3 R11 violations (Missing ICL instance for core
//         which uses IJTAG).
// Error: Instrument instance 'picccpu_maxlen16_edt_i'
//         (core 'picccpu_maxlen16_edt', instrument type 'edt')
//         uses IJTAG but has no ICL instance corresponding to it in
//         the ICL model. Rerun ICL extraction and provide the missing
//         ICL model. (R11-1)
```

R12

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies that core-level iCalls are at the end of the test_setup procedure or the start of the test_end procedure.

The tool automatically maps all iCalls located at the end of the test_setup procedure or at the beginning of the test_end procedure; all others are ignored.

By default this violation is a warning and can be changed to error or note. If you downgrade this DRC from error and has a violation, the IJTAG retargeting capability skips all core-level iCalls that have an invalid position.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'setup'
//         of ICL instance 'piccpu_maxlen16_edt_i' (ICL module
//         'piccpu_maxlen16_edt') in test_setup,
//         but not at the end of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-1)
// Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'my_proc'
//         in test_end, but not at the beginning of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-2)
// Error: There were 2 R12 violations (Position of IJTAG in core-level
//         test procedure prohibits automatic retargeting).
```

R13

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that the used core-level iProcs are loaded in memory and are registered against the right ICL instances.

By default, this violation is an error and can be downgraded to warning or note. If this DRC is downgraded from an error and has a violation, the IJTAG retargeting functionality skips all core-level iProc calls that have not been read in at the top-level.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'setup'
//         in test_setup, but this iProc is not read for ICL module
//         'piccpu_maxlen16_edt'. (R13-1)

// Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'my_proc'
//         in test_end, but this iProc is not read for ICL module
//         'piccpu_maxlen16'. (R13-2)

// Error: There were 2 R13 violations (Missing iProc used at core level).

// Error: Core 'piccpu_maxlen16_edt' (instrument type 'edt') calls iProc
//         'setup', but this iProc was not read for ICL module
//         'piccpu_maxlen16_edt'. (R13-1)
```

R14

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error (severity can be changed to warning)

report_drc_rules: Supported

Validates mapping of static and dynamic control signals.

This DRC validates that static and dynamic EDT IP control signals are properly mapped to the top. Any IJTAG-controlled instruments that do not have a scan interface are also validated.

If the value on the IP boundary is different than what is expected or the signal cannot be traced to the top, the tool issues an R14 violation.

By default, this violation is an error and can be downgraded to warning. When R14 fails and is downgraded to a warning, no automatic procedure updates, clock and constraint adding are performed for the failing pin. You must ensure pin requirements in this case.

This DRC reports the following possible error messages:

```
// Error: Tracing with the stable_shift simulation context from
// core-level control input pin
// 'TOP_DE1_rtl_tessent_occ_elo_inst/slow_clock'
// (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
// to the top level has failed. (R14-1)

// Error: Tracing with the stable_capture simulation context from
// core-level control input pin
// 'TOP_DE1_rtl_tessent_occ_elo_inst/slow_clock'
// (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
// to the top level has failed. (R14-1)

// Error: Core-level control input pin
// 'TOP_DE1_rtl_tessent_occ_elo_inst/capture_cycle_width[1]'
// (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
// is not driven to the expected value of '0'
// or tracing with the stable_shift simulation context to the
// top level has failed.
// Simulated pin value is '1'. (R14-2)

// Error: Tracing with the stable_shift simulation context from
// core-level control input pin
// 'cba9_gate_tessent_lpct_i/scan_en'
// (core 'cba9_gate_tessent_lpct', instrument type 'lpct')
// to the top level has failed.
// Multiple potential drivers found:
//   'tie_high' (31.0),
//   'CS' (5.0).
// Use 'set_attribute_value -name constraint_value_during_load_unload' /
// command to constrain ports and guide the tracer. (R14-1)

// Error: Tracing with the stable_capture simulation context from
// core-level control input pin
// 'cba9_gate_tessent_lpct_i/scan_en'
// (core 'cba9_gate_tessent_lpct', instrument type 'lpct')
// to the top level has failed.
// Multiple potential drivers found:
//   'tie_high' (31.0),
//   'CS' (5.0).
// Use the 'add_input_constraints' command to constrain ports
// and guide the tracer. (R14-1)
```

R15

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates consistency of static and dynamic signals.

This DRC validates that static and dynamic EDT IP signals have consistent requirements properly mapped to the top.

The R15 rules checks if any top-level ports have the same requirements between multiple instruments and already specified values (clock-off state, constrain value).

This DRC checks the following properties:

- Is clock port offstate is correct
- Is the capture constraint value correct
- Is the load_unload force value correct
- Is the shift force value correct

All channel sharing rules are validated by the existing K15 rule.

By default R15 handling is set to error and cannot be downgraded.

This DRC reports the following possible error messages:

```
// Error: Conflicting top-level clock 'edt_clock' clock-off values
// between core instances and user defined clock. (R15-1)
// The clock-off value defined at the top-level is '1'.
//
//      Core Pin     Instance     Core      Type   Top Level   Core Level
//      -----  -----  -----  -----  -----
//      edt_clock  blk1_edt_i  blk1_edt  edt      0          0
//      edt_clock  blk2_edt_i  blk2_edt  edt      0          0
//      edt_clock  blk3_edt_i  blk3_edt  edt      0          0
//      edt_clock  blk4_edt_i  blk4_edt  edt      0          0
//      edt_clock  blk5_edt_i  blk5_edt  edt      0          0
//      edt_clock  blk6_edt_i  blk6_edt  edt      0          0
//      edt_clock  blk7_edt_i  blk7_edt  edt      0          0
//
// Error: Conflicting top-level clock 'edt_clock' clock-off values
// between core instances. (R15-2)
// Expected clock-off value is '1'.
//
//      Core Pin     Instance     Core      Type   Top Level   Core Level
//      -----  -----  -----  -----  -----
//      edt_clock  blk1_edt_i  blk1_edt  edt      1          1
//      edt_clock  blk2_edt_i  blk2_edt  edt      0          0
//      edt_clock  blk3_edt_i  blk3_edt  edt      0          0
//      edt_clock  blk4_edt_i  blk4_edt  edt      0          0
//      edt_clock  blk5_edt_i  blk5_edt  edt      0          0
//      edt_clock  blk6_edt_i  blk6_edt  edt      0          0
//      edt_clock  blk7_edt_i  blk7_edt  edt      0          0
//
// Error: Conflicting top-level port 'edt_low_power_shift_en' forces
// requirement in 'load_unload' procedure between core instances.
// (R15-3)
// Expected force value is '1'.
//
//      Core Pin           Instance     Core      Type   Top Level   Core Level
//      -----  -----  -----  -----  -----
//      edt_low_power_shift_en  blk1_edt_i  blk1_edt  edt      1          1
//      edt_low_power_shift_en  blk2_edt_i  blk2_edt  edt      0          0
//      edt_low_power_shift_en  blk4_edt_i  blk4_edt  edt      1          1
//      edt_low_power_shift_en  blk7_edt_i  blk7_edt  edt      1          1
//
// Error: Conflicting top-level port 'edt_update' forces requirement in 'shift'
// procedure between core instances. (R15-4)
// Expected force value is '0'.
//
//      Core Pin     Instance     Core      Type   Top Level   Core Level
//      -----  -----  -----  -----  -----
//      edt_update  blk1_edt_i  blk1_edt  edt      0          0
//      edt_update  blk2_edt_i  blk2_edt  edt      0          0
//      edt_update  blk4_edt_i  blk4_edt  edt      1          0
//      edt_update  blk7_edt_i  blk7_edt  edt      1          0
//
// Error: Conflicting top-level port 'edt_clock' capture constraint values
// between core instances and user defined constraint. (R15-5)
// The capture constraint value defined at the top-level is '1'.
//
```

```

//          Core Pin   Instance   Core      Type   Top Level   Core Level
//          -----  -----  -----  -----  -----  -----
//          edt_clock blk1_edt_i blk1_edt  edt     0           0
//          edt_clock blk2_edt_i blk2_edt  edt     0           0
//          edt_clock blk3_edt_i blk3_edt  edt     0           0
//          edt_clock blk4_edt_i blk4_edt  edt     0           0
//          edt_clock blk5_edt_i blk5_edt  edt     0           0
//          edt_clock blk6_edt_i blk6_edt  edt     0           0
//          edt_clock blk7_edt_i blk7_edt  edt     0           0
//
//  Error: Conflicting top-level port 'edt_clock' capture constraint values
//         between core instances. (R15-6)
//         Expected capture constraint value is '1'.
//
//          Core Pin   Instance   Core      Type   Top Level   Core Level
//          -----  -----  -----  -----  -----  -----
//          edt_clock blk1_edt_i blk1_edt  edt     1           0
//          edt_clock blk2_edt_i blk2_edt  edt     0           0
//          edt_clock blk3_edt_i blk3_edt  edt     0           0
//
//  Error: There were 6 R15 violations (Conflicting port requirements).

```

R16

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates EDT IP core scan chain binding.

This DRC validates if the scan chain binding is correct for the newly added EDT IP cores. This DRC is executed after automatic scan chain binding will be executed.

The tool attempts to bind EDT instruments with the following user-defined scan chains:

- Direct access chains
- User added internal chains that are not bound to an EDT block

Direct access chains can be bound with an EDT instrument in bypass or single bypass mode.

This DRC reports the following possible error messages similar to the following:

If corresponding EDT instance scan pins are connected to different chains:

```
// Error: Core instance 'core_1/picccpu_maxlen16_1_edt_i'  
// (core 'picccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)  
// EDT instance scan pins corresponding to chain index 1 are connected to two  
// different scan chains:  
// EDT scan pin 'core_1/picccpu_maxlen16_1_edt_i/edt_scan_out[0]'  
// connects to scan chain input 'core_1/input56' of 'chain56'  
// EDT scan pin 'core_1/picccpu_maxlen16_1_edt_i/edt_scan_in[0]'  
// connects to scan chain output 'core_1/output57' of 'chain57'
```

If one side of the chain is bound, and the pin specified for the other side does not exist in the design (so not a connectivity issue):

```
// Error: Core instance 'core_1/picccpu_maxlen16_1_edt_i'  
// (core 'picccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)  
// One of the scan pins of the following scan chains does not exist in the  
// design:  
// chain56 (the scan output 'core_1/picccpu_maxlen16_1_edt_i/edt_scan_out1[1]'  
// does not exist in the design),  
// chain57 (the scan input 'core_1/foo' does not exist in the design).
```

If one side of the chain is bound, and the other side has a valid pin with no binding:

```
// Error: Core instance 'core_1/picccpu_maxlen16_1_edt_i'  
// (core 'picccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)  
// One of the scan pins of the following scan chains is not connected to the EDT  
// instance:  
// chain56 (no connection found from scan chain outputABC' to the EDT  
// instance pin 'core_1/picccpu_maxlen16_1_edt_i/edt_scan_out[1]'),  
// chain57 (no connection found from EDT instance pin  
// 'core_1/picccpu_maxlen16_1_edt_i/edt_scan_in[3]' to scan chain input  
// 'core_1/foo').
```

If neither side is bound, and both scan pins do not exist in the design:

```
// Error: The scan pins of the following scan chains do not exist in the design.  
// (R16-1)  
// edt_chain0 (input = 'foo1', output = 'bar1'),  
// edt_chain1 (input = 'foo2', output = 'bar2').
```

If neither side is bound, and SCI exists in the design but SCO does not:

```
// Error: The scan output pins of the following scan chains do not exist in the  
// design and their scan input pins do not connect to any EDT instance. (R16-1)  
// edt_chain0 (input = 'input1', output = 'foo'),  
// edt_chain1 (input = 'input2', output = 'bar').
```

If neither side is bound, and SCO exists in the design but SCI does not:

```
// Error: The scan input pins of the following scan chains do not exist
// in the design and their scan output pins do not connect to any EDT
// instance. (R16-1)
//   edt_chain0 (input = 'foo', output = 'output1'),
//   edt_chain1 (input = 'bar', output = 'output2').
```

If neither side is bound, but both SCI and SCO exist in the design:

```
// Error: The scan pins of the following scan chains do not connect to any EDT
// instance. (R16-1)
//   edt_chain0 (input = 'input1', output = 'output1'),
//   edt_chain1 (input = 'input2', output = 'output2').
```

R17

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates if EDT IP unused channels are tied off.

Accessible input channel number selection requires that unused channels are feeding constant '0' values into the decompressor.

By default R17 handling is set to error and cannot be downgraded.

This DRC reports the following possible error messages:

```
// Error: Core instance 'core_1/picccpu_maxlen16_1_edt_i' (core
//        'picccpu_maxlen16_1_edt', instrument type 'edt') (R17-1)
//        has inaccessible input channels that are not tied off:
//        core_1/picccpu_maxlen16_1_edt_i/edt_channels_in[2],
//        core_1/picccpu_maxlen16_1_edt_i/edt_channels_in[3].
// Error: Core instance 'core_1/picccpu_maxlen16_1_edt_i' (core
//        'picccpu_maxlen16_1_edt', instrument type 'edt') (R17-2)
//        has an inaccessible input channel that is not tied off:
//        core_1/picccpu_maxlen16_1_edt_i/edt_channels_in[3].
// Error: There were 2 R17 violations (Inaccessible input channels
// not tied off).
```

R18

Category: Tessent OCC

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies after scan chain tracing that the Tessent OCC registers are part of the scan chains.

Applies only when using the Tessent OCC in a mode where it is controllable by scan.

This DRC reports the following possible error messages:

```
// Error: OCC configuration registers of core instance
// 'top_gates_tessent_occ_NX1_inst' (core 'top_gates_tessent_occ',
// instrument type 'occ') are not part of active scan chains. (R18-1)
// The following OCC configuration registers exist in the design but
// are not part of any active scan chain.
//   'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_0_',
//   'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_1_',
//   'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_2_',
//   'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_3_'.
// Please make sure all configuration registers of OCCs that are active
// in this mode are stitched into scan chains that are active in this
// mode and all EDT instances active in this mode were added using the
// 'add_core_instances' command.
// Error: There was 1 R18 violation (OCC configuration registers are not
// part of any active scan chains).

// Error: OCC configuration register of core instance
// 'top_gates_tessent_occ_NX1_inst' (core 'top_gates_tessent_occ',
// instrument type 'occ') does not exist in the design. (R18-1)
// Please make sure the hierarchy inside the OCC IP is preserved during
// synthesis, and that design object names are not changed.
//   'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/KUKU[0]'.
// Error: There was 1 R18 violation (OCC configuration registers are not
// part of any active scan chains).
```

R19

Category: Tessent OCC

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies if Tessent OCC's internal nodes exist in the design.

This DRC reports the following possible error message:

```
// Error: Core '<OCC IP instance>' (core '<OCC IP core name>', instrument
type 'occ') (R19-1)
// The following OCC internal paths do not exist in the design.
// /top_gates_tessent_occ_NX1_inst/dummy_path1,
// /top_gates_tessent_occ_NX1_inst/dummy_path2.
```

R20

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates that if an iCall is retargeted from one level to the next, then this iCall is for an iProc and not an iTopProc.

Detects if an iTopProc was used at the core level. It is not triggered for core instances added with -current_design switch.

By default R20 handling is set to error and can be downgraded to a warning or a note. If the DRC is downgraded, those iCalls are not mapped.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16' (mode 'internal') calls iTopProc
//         'my_proc' in the test_end procedure.
//         iTopProc can only be defined for the top level and therefore
//         cannot be retargeted from a lower-level instance. (R20-1)
// Error: There was 1 R20 violation (iTopProc used at core level).
```

R22

Category: EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies instrument IJTAG requirements for instruments without a scan interface.

This DRC can be changed to error or note. R22 typically applies to scan instruments like the EDT controller where static control signals like edt_bypass are declared as DataIn ports.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is an
// IJTAG instrument with no scan interface, but top-level ICL was not
// extracted or read.
// Use the 'extract_icl' command to extract top-level ICL or read the
// top-level ICL model. (R21-2)
// Error: There was 1 R22 violation (Top-level ICL not extracted while
// IJTAG instruments with no scan interfaces present in the design).

// Warning: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is
// an IJTAG instrument with no scan interface, but top-level ICL was not
// extracted or read.
// The PDL will not be automatically appended to the top-level
// procedure. (R22-1)
// Warning: There was 1 R22 violation (Top-level ICL not extracted
// while IJTAG instruments with no scan interfaces present in the design).
```

R23

Category: EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed to a Warning or Note)

report_drc_rules: Supported

Verifies scan instrument IJTAG requirements.

This DRC can be downgraded to a Warning or Note.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is an
// IJTAG instrument with scan interface, but top-level ICL was not
// extracted or read.
// Use the 'extract_icl' command to extract top-level ICL or read the
// top-level ICL model. (R23-1)
// Error: There was 1 R23 violation (Top-level ICL not extracted
// while IJTAG instruments with scan interfaces present in the design).
```

R27

Category: OCC Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that the active OCCs are not driven by the outputs of other active OCCs during capture.

Cascading of OCCs can lead to incorrect capture clock sequences and cause mismatches. Active OCCs in the parent mode driving active child OCCs will not cause a R27 violation. This DRC will be executed during ATPG and pattern retargeting step.

This DRC reports the following error message:

```
// Error: Capture clock 'corec_i1/corec_rtl2_tessent_occ_clka_inst/fast_clock'
//          (20442.1) of OCC instance 'corec_i1/corec_rtl2_tessent_occ_clka_inst'
//          (core 'corec_rtl2_tessent_occ') is driven by OCC instance
//          'coreb_i1/corea_i1/corea_rtl2_tessent_occ_clka_inst'
//          (core 'corea_rtl2_tessent_occ') output clock 'coreb_i1/corea_i1/
//          corea_rtl2_tessent_occ_clka_inst/clock_out' (19140.0).
//          An active OCC cannot be driven by another active OCC unless the driven
//          OCC is a child OCC and the driving OCC is operating in parent mode.
//          This may lead to incorrect OCC capture clock sequences and result in
//          mismatches. (R27-1)
```

R28

Category: OCC Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting, and patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Verifies that, in fast_capture mode, the fast_clock ports of OCC instances are driven by a pulse-always, pulse-in-capture, or an asynchronous clock.

This DRC performs the following checks:

1. The OCC fast clock port must be driven by a port or a pseudo-port.

Example:

```
// Warning: Fast clock pin 'coreb_rtl2_tessent_occ_clka_inst/
fast_clock' (1990.1)
//          of OCC instance 'coreb_rtl2_tessent_occ_clka_inst'
(core 'coreb_rtl2_tessent_occ')
//          is not driven by a clock.
//          This may result in incorrect patterns. (R28-1)
```

2. The OCC fast clock port driver, which is a port or pseudo-port, must be defined as a clock.

```
// Warning: Fast clock pin 'coreb_rtl2_tessent_occ_clkc_inst/fast_clock' (1992.1)
//          of OCC instance 'coreb_rtl2_tessent_occ_clkc_inst' (core
'coreb_rtl2_tessent_occ')
//          is driven by port 'clkc' (3.0) which is not defined as a clock.
//          This may result in incorrect patterns. (R28-2)
```

3. The OCC fast clock driver must be a clock that is pulse-always, pulse-in-capture, or an asynchronous clock.

```
// Warning: Fast clock pin 'coreb_rtl2_tessent_occ_clkb_inst/
fast_clock' (1991.1)
//          of OCC instance 'coreb_rtl2_tessent_occ_clkb_inst' (core
'coreb_rtl2_tessent_occ')
//          is driven by clock 'clkb' (2.0) which is not pulse-
always,
pulse-in-capture, or asynchronous (defined with a period).
//          This may result in incorrect patterns. (R28-3)
```

Use the [set_drc_handling](#) command to change the handling of this DRC to error or note.

Scannability Rules (S Rules)

Scan substitution and stitching is done with respect to library cells and pre-existing scan segments. For library cells, a simple cell may just contain a single memory element (_dff primitive in the Tessent cell library) that exists in both the non-scan version and the scan equivalent. In this case, S-rule checking must be performed on that memory element. A more complex library cell may contain some memory elements that will not become part of the scan path after the non-scan model has been replaced with the scan model. S-rules are only run on those memory elements that will become part of the scan path. The scan model for a complex library cell may contain multiple scan input and scan output pins. In this case, the tool will identify the clocks that control these embedded scan segments, and target only those clocks during S-rule analysis. Finally, you can specify pre-stitched scan segments on arbitrary Verilog modules. In this case, S-rules are once again performed with respect to the clock and set/reset pins that you specified as part of the add_scan_segments command. The specific memory elements (dff primitives) that will become part of the scan path after scan chain stitching is complete will be referred to as scan path memory elements.

For each sequential element in the design, the tool performs two main checks. The first check, S1, ensures that when all defined clocks — including sets and resets — are at their off-states, the sequential elements remain stable and inactive. The second check, S2, ensures that for each defined clock, sequential elements can capture data when all the other defined clocks are off. These scannability checks determine if the tool can turn off all set and reset lines, and turn on and off all clock inputs of sequential cells from the design's primary input pins. Without this controllability, the tool will not allow a sequential element to pass scannability checks, which is a requirement to be considered for scan identification.

This checking is similar to the C1 (S1) and C7 (S2) rules checks, which the tool performs to determine the stability of defined scan chains. The C1 and C7 rules perform these same checks on sequential elements that are already converted to scan. For more information on C1 and C7, refer to “[Clock Rules \(C Rules\)](#)” on page 2240.

Once these scannability rules pass, the elements are considered “scannable” and the DRC process treats the non-scan elements as though they were scan elements for the remainder of the DRC rules.

The default handling for S1 and S2 rules is “error”. This identifies the flops that cannot be used for scan insertion due to clock controllability issues and aborts the system mode transition. This allows you to debug your design at this stage instead of discovering it later during ATPG. The default handling can be changed to “warning” with the set_drc_handling command, in which case the flops will remain non-scan and may lead to poor ATPG results.

The default handling for the S3 rule is “warning” which causes the tool to add the necessary pin constraints to allow activating the clock during scan shifting. The handling can be changed to “error”, but this does not abort the system mode transition. Instead, the flops will be inserted into scan chains, but no extra pin constraints will be added.

Tip

i In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see “[How to Report Gate Data](#)” on page 2648 on page 2215.

S1	2985
S2	2986
S3	2987
S4	2988
S5	2989
S6	2990
S7	2991
S8	2992

S1

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S1 checks all the clock inputs (including sets and resets) of each scannable non-scan memory element to ensure that these inputs can be turned off. This rule ensures that non-scan elements that may be converted to scan can be controlled to hold their current data. You can use “report_drc_rules s1” to get a listing of all the violations. This report will include the pin-path and gate-id of the flop with the violation (for scan segments it will report the segment name instead).

To debug an S1 DRC violation without DFTVisualizer, use the command “[analyze_drcViolation <violation_number>](#)”. This will run the DRC analysis for the specific violation and allow the simulation data to be displayed. In the following example, the failure is due to the clock pin “CP” being set to X. To pass the S1 DRC, the clock must be set to its off state of 0. The command sequence noted will work with or without DFTVisualizer.

```
analyze_drcViolation viol S1-2
// Gate report now set to display_sim_data.
// Creating schematic for 3 instances (0 were compacted).
report_drc_rules s1-2
// Warning: Unstable nonscan cell /AA/OUT_2_reg(32) when all clocks are
off. (S1-2)
report_gates 32
// /AA/OUT_2_reg (32) TIEX
// "I0" I (0) 10-
// "I1" I (0) 27-
// CP I (X) 22-/AA/I_1/out
// "I3" I (X) 13-
// "OUT" O (X) 15-
```

Scannability rule S1 is a modified version of the C1 clock rule, in that it performs the same type of checking on nonscan sequential elements that C1 performs on scan elements. For additional information on the C1 clock rule, refer to “[C1](#)” on page 2692.

You can specify how these potential scan cells are handled by setting S1 to one of the following handling options:

- **Error** — identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning** — identifies cells, reports violations, and marks these cells as non-scan.

Note

 Clocks that are defined as pulse-always cannot be turned off during testing. Therefore, flops driven by pulse-always clocks fail the S1-rule checking.
Note that Tessent Scan does not use pulse-always clocks as test-clocks to repair the clock signals of the flops that fail with any S-rules.”

Here are some examples of S1 DRC violation error messages:

- For flop based violation:

```
// Error: Unstable nonscan cell because clock input '/dff1_inst/
CLK' (552.3) is set to X when all clocks are off. (S1-1)
```

- For subchain based violation:

```
// Error: Unstable nonscan cell because clock input (subchain:
'lff_subchain_1/lff2_u2/SO2', pin: 'CLK') is driven by a pulse-
always clock. (S1-122)
```

S2

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S2 checks all clock inputs (not including sets and resets) of each scan path memory element to see whether they can capture data. This rules ensures that a non-scan cell can capture data using one of the defined clocks. In order to be converted to scan, a non-scan cell must be able to capture data when a single clock is on. You can use “report_drc_rules s2” to get a listing of all the violations. This report will include the pin-path and gate-id of the flop with the violation (for scan segments it will report the segment name instead)

Scannability rule S2 is a modified version of the C7 clock rule, in that it performs the same type of checking on non-scan sequential elements that C7 performs on scan elements. For additional information on the C7 clock rule, refer to “[C7](#)” on page 2716.

You can specify how these potential scan cells are handled by setting S2 to one of the following handling options:

- **Error** — identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning** — identifies cells, reports violations, and marks these cells as non-scan.

Here is an example of an S2 DRC violation error message:

```
// Error: Clock capture ability check failed at clock input '/dff1_inst/CLK' (553.3) (S2-1)
```

S3

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Warning

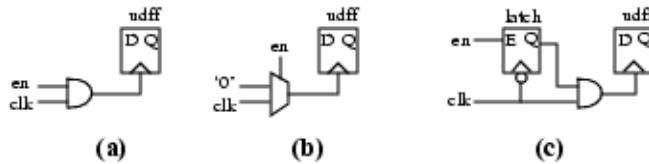
report_drc_rules: Supported

All non-clock primary input pins in the clock cone of scan path memory elements must be constrained. If these pins are unconstrained and made scannable, trace violations may occur.

Non-clock primary input pins in the clock cone of non-scan memory elements should be constrained in the ATPG dofile or load_unload procedure to correctly sensitize the clock path.

[Figure 7-51](#) illustrates S3 violations where the pin ‘clk’ is defined as the shift clock with an off state value of ‘0’. The non-clock pin ‘en’ causes the violation if it is left un-constrained.

Figure 7-51. Example S3 Rule Violation



The total number of S3 violations are reported in the session transcript during the DRC rule checking. Individual S3 violations can be reported with the report_drc_rules and report_scan_elements commands.

The default DRC handling of the S3 rule is “Warning,” which means that memory elements with S3 violations are kept in the pool of scannable candidates. The handling can be changed to “Error” using the set_drc_handling command, in which case the memory elements with S3 violations are excluded from the pool of scannable candidates automatically.

If you want to include some of those memory elements back in scan insertion, you can issue delete_nonscan_instances commands on them. Similarly, if the S3 rule handling is “Warning” (the default), you can obtain the list of the violations using the report_scan_elements or report_drc_rules commands and issue add_nonscan_instances commands on a subset of the list explicitly.

Although the handling of the S3 DRC violation can be changed to “Error,” the effect of the violation is different from that of the S1 and S2 rules. The tool does not insert_test_logic on the clock pins to correct an S3 violation even if the user turns on the test logic insertion using the set_test_logic command.

Here is an example of an S3 DRC violation error message:

```
// Warning: Un-initialized non-clock input detected for clock port aa/udff2/CLK
// Primary input pin "d" must be set to 0 (S3-1)
```

S4

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Note

report_drc_rules: Supported

Scannability rule S4 identifies and determines the handling of non-scan memory elements that satisfy certain conditions.

These conditions are:

- Meet requirements for use as scan elements.
- Maintain constant output values during the shift and capture cycles.

- Are in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.
- Contribute to the data lines of the memory elements in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.

Note

 Flip-flop/latch elements that maintain constant output values during shift and capture cycles can potentially disturb the clock and set/reset signals of other flip-flops/latches, RAMs, and ROMs if their constant output values are not maintained.

If non-scan cells that maintain constant output values and are associated with clock and set/reset signals are converted to scan cells, they can prevent the clock and set/reset signal paths from being sensitized properly. When this occurs, more test logic may be required to sensitize the clock path.

You can specify how these non-scan cells are handled by setting S4 to one of the following handling options:

- **Note** — reports the non-scan cells to the log file and leaves them as non-scan cells. Default handling.
- **Warning** — reports the non-scan cells to the log file and converts them to scan cells. The resulting scan cells will have an unknown value (TIEX) for the output during the remainder of the DRC.
- **Ignore** — allows non-scan cells to be converted to scan cells; no reporting is done.

You can report on the S4 results with the `report_drc_rules` command.

Here is an example of an S4 DRC violation error message:

```
// Note: Non-scan cell /dff1(14) that maintains a constant value during capture and shift phases will remain non-scan. (S4-1)
```

S5

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S5 identifies clock sources that drive set/reset ports as well as clock ports.

When a clock source (a top-level pin or an internally defined clock net) drives the set/reset port of one or more flip-flops while also driving the clock port of one or more flip-flops, the set/reset port driven by this signal is reported as an S5 violation.

You can optionally fix S5 violations by adding test logic at the set/reset ports driven by the clock. The tool will then treat the set/reset ports as uncontrollable and insert the same test logic that is used to fix the set/reset ports of flip-flops with an S1 violation.

You can specify how these potential scan cells are handled by setting S5 to one of the following handling options:

- **Error** — identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning** — identifies cells, reports violations, and marks these cells as non-scan.

Here is an example of an S5 DRC violation error message:

```
// Error: Set/reset port (subchain: '/my_inst/TEST__SOUT[4]', pin: 'R0') is driven by clock
// '/test_TEST__SHIFT_CLK1' (36)

// This signal also drives clock port (subchain: '/my_inst/TEST__SOUT[0]', pin:
// 'TEST__SHIFT_CLK[1]') which must pulse during shift (S5-1)
```

S6

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error (in dft -scan context); Note (in dft -test_points context)

report_drc_rules: Supported

Scannability rule S6 identifies both subchains and pre-existing scan cells (that are not part of declared scan chains or subchains) that have driven scan-in ports and reports them as an S6 violation.

Subchain instances with driven scan-in ports are deleted. That is, they are no longer considered subchains.

By default, an S6 violation is reported as an Error, the system mode remains as setup, and the following error message displays:

```
// Error: There were <int> scannability rule S6 fails (driven scan-in pin
of a sub chain or of a scan cell in unspecified scan chain).
```

You can optionally choose to change the handling of this rule to Warning. In this case, the tool allows the system mode to change to analysis, but deletes subchain instances that violate the S6 rule and ignores scan cells that violate the S6 rule when stitching into scan chains during the scan chain generation process. The following warning message displays:

```
// Warning: There were <int> scannability rule S6 fails (driven scan-in
pin of a sub chain or of a scan cell in unspecified scan chain).
```

You can optionally choose to change the handling of this rule to Note. In this case, the tool allows the system mode to change to analysis, but deletes subchain instances that violate the S6 rule; however, scan cells that violate the S6 rule will still be treated as scannable cells. The following warning message displays:

```
// Note: There were <int> scannability rule S6 fails (driven scan-in pin of a sub chain or of a scan cell in unspecified scan chain).
```

You can use the [report_drc_rules](#) command to list the details of an S6 rule violation.

When scan cells produce S6 violations, you can optionally fix them by explicitly declaring the scan chains or subchains that the identified scan cells belong to using the [add_scan_chains](#) command. The tool will use this information to validate the traceability of declared scan chains or subchains when a test procedure file containing `load_unload` and `shift` procedures is present.

You can optionally declare the scan cells violating the S6 rule to be non-scan instances using the [add_nonscan_instances](#) command. In this case, the tool ignores these cells when stitching scan chains during the scan chains generation process.

S7

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Note

report_drc_rules: Supported

Scannability rule S7 identifies potential scan cells with a constant data input.

During scan insertion, memory elements with input signals tied to 0 or 1 are treated as scannable cells, which improves stuck-at fault coverage. However, these paths may not be optimized for timing, and are not used in functional mode. By default, all cells with constant input values are identified but are not excluded from scan insertion unless the constant value is in the clock control logic (that is, the output of the flop is in the fan-in cone of a clock or set/reset port of another flop).

Only the data inputs are checked for a constant 1 or 0 value. Cells that are “constant” because the set/reset signal is forced active are not considered, but those gates would already be marked as non-scan because of the implied S1 rule violation.

Note that tied values do not propagate through sequential elements unless a `test_setup` procedure is used to pulse the relevant clocks.

You can specify how these potential scan cells are handled by setting S7 to one of the following handling options:

- **Error** — identifies cells with constant data inputs, and stops after identifying all DRC violations.
- **Warning** — identifies cells, reports violations, and marks these cells as non-scan.
- **Note** — identifies cells, reports violations, but allows these cells to become scan cells. This is the default DRC handling of rule S7. Maintains backward compatibility.
- **Ignore** — does not check for constant data input.

For each DRC violation, the instance name of the scan cell, the affected data-input pin, and the constant value that was identified will be reported. You may use the analyze_drcViolation command to graphically display the failing gate along with the sensitization values that led to the violation. This can be used to trace backward from the data input pin of the flop and identify the source of the constant value that triggered this violation.

S8

Category: Scannability

Contexts Supported: dft -scan

Default Handling: Error

report_drc_rules: Supported

Scannability rule S8 identifies the OCC circuits that control the clocks for all of the scan elements in any scan mode, and then it verifies that the OCC control scan segments that are needed to operate the clocks for the scan elements within in each scan mode are also part of the population for that mode.

When an OCC segment is omitted from a scan mode population that also includes some of scan elements clocked by its OCC, it is reported as an S8 violation. By default, an S8 violation is reported as an Error, and the following error message displays:

```
// Error: There was 1 S8 violation (OCC scan segment not included in a
scan mode's population that contains scan elements clocked by its OCC).
```

You can optionally choose to change the handling of this rule to Warning using the set_drc_handling command. In this case, the following warning message displays:

```
// command: set_drc_handling s8 warn
// command: analyze_scan_chains
.....
.....
.....
// Warning: There was 1 S8 violation (OCC scan segment not included in a
scan mode's population that contains scan elements clocked by its OCC)
....
```

If the OCC segment is not part of the population because it marked as is_non_scannable, then this is reported in the error message:

```
// Error: OCC scan segment '/corea_rtl_tessent_sib_sti_inst/ltest_so' is
not included in the population of the unwrapped scan mode 'unwrapped'
which contains scan elements clocked by its OCC.
// Note that the OCC scan segment '/corea_rtl_tessent_sib_sti_inst/
ltest_so' is non scannable for this reason: user_specified. (S8-1)
```

Scan Chain Trace Rules (T Rules)

Using the information in the test procedure files, the rules checker traces the scan chains to identify the scan cells and all memory elements associated with the scan cells. It then classifies the scannable memory elements as either MASTER, SLAVE, SHADOW, COPY, or EXTRA. Violations of scan chain trace rules are error or warning conditions, and you cannot change the handling of these rules—with the exception of rule T18, which you can set to ignore.

The following subsections describe each of the trace rules.

Tip In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

T1.....	2995
T2.....	2995
T3.....	2996
T4.....	3000
T5.....	3005
T6.....	3007
T8.....	3008
T9.....	3008
T10.....	3009
T11.....	3009
T12.....	3009
T13.....	3010
T14.....	3010
T15.....	3010
T16.....	3011
T17.....	3011
T18.....	3012
T19.....	3012
T20.....	3013
T21.....	3013
T22.....	3013
T23.....	3014
T24.....	3014
T25.....	3015

T26.....3016

T1

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

All defined scan chains must contain at least one scan cell. Correct this error condition by deleting the indicated scan chains.

The error message is:

```
No scan cells identified in scan chain C. (T1-1)
```

C is the scan chain name.

T2

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A scannable memory element can reside in more than one scan chain unless you have issued the “set_drc_handling -ALLOW_COMMON_SCAN_CELLS_IN_MULTIPLE_CHAIN OFF” command.

However, there are some restrictions on scan cell sharing, which result in a T2 violation when not observed:

- Scan cell sharing is not allowed for compressed chains driven by a decompressor.
- The shared scan cells must be at the logical scan cell boundary. A library model with multiple scan cells cannot be partially shared. In other words, you cannot have the merged point within the state elements of a logical scan cell.
- The shared scan cells must not end at a master segment of a master/slave combination.

You can display the complete paths of the scan chains by using the set_trace_report command by turning trace reporting on and then repeating the rules checking. If you use all scan chains, you may need to make netlist modifications to correct this error condition.

The T2 DRC will flag a violation when the split point drives scan cells with different clock edges. The split point and the driven point need to be at the same clock edge, for example leading edge to leading edge, trailing edge to leading edge, or trailing edge to trailing; never leading edge to trailing edge.

The T2 rule has three possible error messages, in which N is an instance name, and G is its gate ID number:

- You have issued the “set_drc_handling -ALLOW_COMMON_SCAN_CELLS_IN_MULTIPLE_CHAIN OFF” command, and the scan cell exists in more than one scan chain:

N (G) already used in chain trace. (T2-1)

- Two scan chains share a merge point that is within a single library model:

Chain N and N split at N (G) which is not a logical cell boundary.
(T2-1)

- The scan cell exists in more than one scan chain, and at least one of the chains is compressed:

Scan cell N (G) is shared by compressed chains N and N. (T2-1)

T3

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **shift** procedure must create a sensitizable path from the scan chain output back to the scan chain input. An improperly sensitized gate in the scan path will cause an error condition, which the tool reports as a T3 DRC violation.

The occurrence message is:

Scan chain S blocked at gate N (G) after tracing C cells. (T3-1)

S is the scan chain name, N is the instance name, G is its gate ID number, and C is the number of scan cells traced in the scan chain.

The summary message is:

T3 : #fails=N handling=error (scan path blocked)

N is the number of T3 rule violations.

How to Debug T3 Violations in DFTVisualizer

You can also debug T3 violations in the DFT visualizer. When the tool encounters the error, it will report this message:

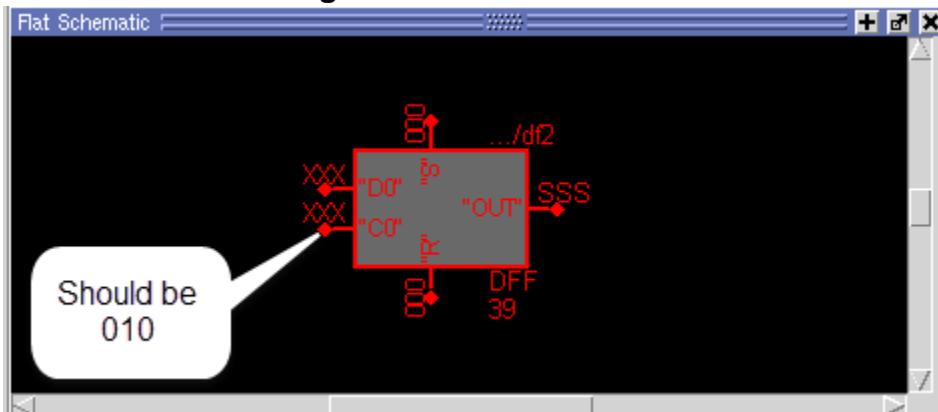
```
// Error: Scan chain chain1 blocked at gate /reg2/df2 (39) after tracing  
1 cells. (T3-1)  
// Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

You can now analyze the drc error in DFTVisualizer using this command:

analyze_drcViolation T3-1

DFTVisualizer starts and will display /reg2/df2 in the Flat Schematic window as shown in [Figure 7-52](#).

Figure 7-52. T3 Violation



The clock “C0” should be 010, not XXX. Trace backwards by clicking on the pin for “C0”.

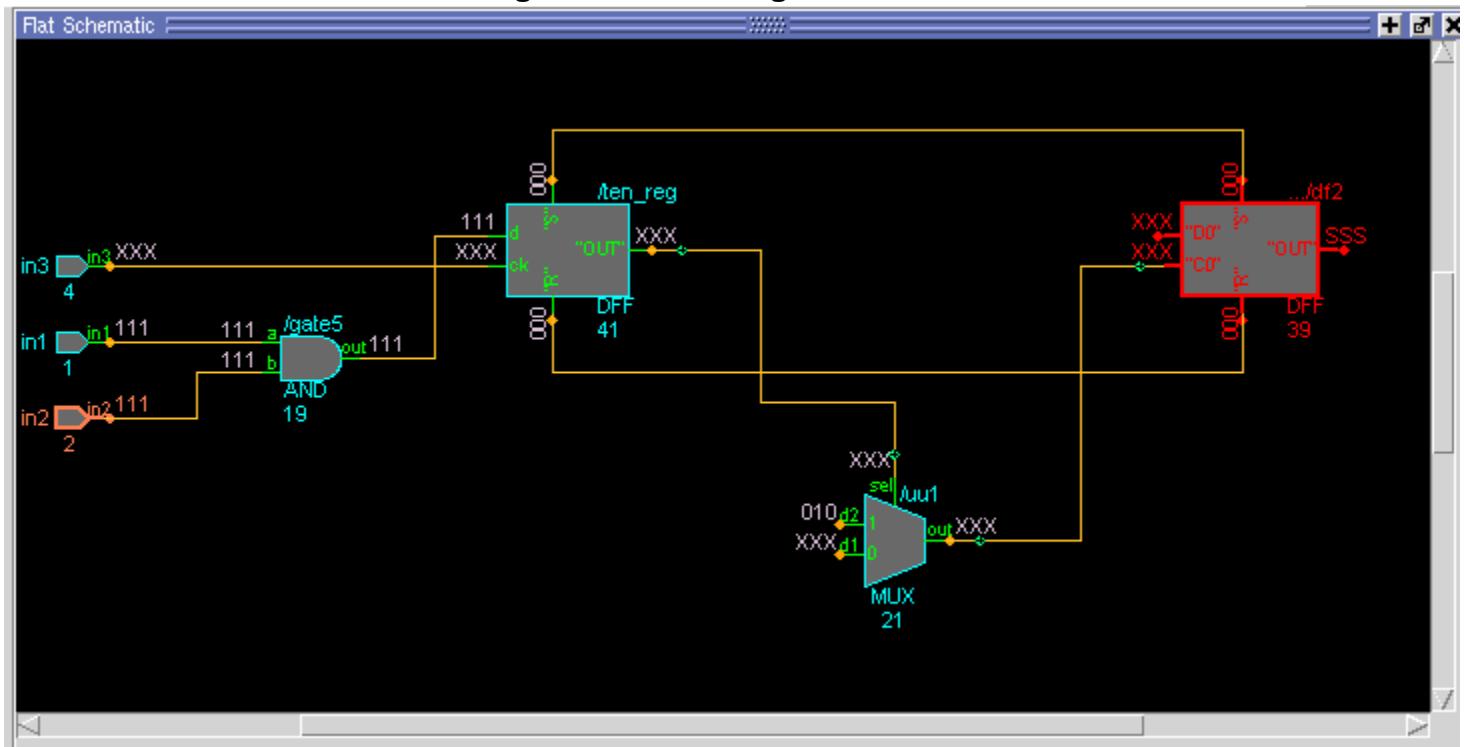
Note

When a T3 violation occurs, and you use the “analyze_drc_violation” command, the gate report is set to trace and the report will show an SSS on some paths. The S distinguishes scan path “don’t cares” from non-scan path “don’t cares”, which are shown as X.

The same happens when you used the “set_gate_report drc shift” command.

The results of tracing backwards from “C0” is shown in [Figure 7-53](#).

Figure 7-53. Tracing T3 Violation



The “C0” pin is driven by the output of MUX whose “sel” pin is XXX. That pin is driven by the output of DFF (a flop) whose clock pin, “ck” is driven by a primary input “in3” which is XXX.

The solution is to provide a correct test_setup procedure that initializes the device to the correct mode. In this example, add the the primary input “in3” as a clock and constrain “in1” and “in2”:

```
add_clock 0 in3
add_input_constraint in1 -c1
add_input_constraint in2 -c1
```

Modify the test procedure to pulse “in3”:

```
timeplate gen_tp1 =
    offstate "in3" 0 ;
    force_pi 0 ;
    measure_po 100 ;
    pulse in3 200 100 ;
    pulse clock 200 100 ;
    period 400 ;
end;
...
procedure test_setup =
    timeplate gen_tp1 ;
    cycle =
        force oe 1 ;
        force in1 1 ;
        force in2 1 ;
        force in3 0 ;
        pulse in3;
    end ;
end;
```

How to Debug T3 Violations Without DFTVisualizer

Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. To do this, set the gate reporting to trace, and use the **report_gates** command for the gate ID number displayed in the error message. This can help you to identify where the blockage occurs; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
Error: Scan chain chain1 blocked at gate /reg2/df2 (39) after tracing 1
cells. (T3-1)
```

The message tells you that the scan chain, “chain1”, is blocked at the gate /reg2/df2. The tool stays in SETUP mode. To proceed to debug, set up the tool to be in trace mode. If you have not already done so, issue the following commands:

```
set_trace_report on
set_gate_level primitive
set_gate_report trace
```

Re-run the DRC rule checks by going back into ANALYSIS mode and then report the gate that is identified as having a T3 error.

```
set_system_mode analysis
report_gate /reg2/df2

//  /reg2/df2 (39)  DFF
//      "S"      I  (000)  10-
//      "R"      I  (000)  9-
//      "C0"     I  (XXX)  26-
//      "D0"     I  (XXX)  23-
//      "OUT"    O  (SSS)  13-  14-
```

When you run report_gate with trace on, you expect the scan clock to show “010”, the constrained pins to be at their constrained values, scan enable to be active, and all sets and resets to be inactive. Scan out and scan in (on scan elements) should all be “X”. On all the combinational elements, in the scan chain, you should see “X” on the output and “X” on one of the inputs. When you have “X” on two inputs, the chain is blocked.

In this example, the blocked gate is a flop. Pins “S” and “R” are both inactive, which is okay, but the clock “C0” and data “D0”pins have a problem. “C0” is not shifting, and since this is a flop, it should show (010). Do a backwards trace to find out what drives the clock. In this example, we will set the gate level to be design.

```
set_gate_level design
report_gate /reg2/df2

//  /reg2  dfsc
//    ck    I  (XXX)  /uu1/out
//    d0    I  (XXX)  /gate2/out
//    d1    I  (XXX)  /reg3/qbar
//    sc    I  (111)  /scan_en
//    q     O  (XXX)  /gate3/a
//    qbar  O  (SSS)  /reg1/d1
```

The clock, “ck”, is driven by /uu1/out, so report that gate.

```
report_gates /uu1

//  /uu1  mux21
//    d1    I  (XXX)  /reg1/q
//    d2    I  (010)  /clock
//    sel   I  (XXX)  /ten_reg/q
//    out   O  (XXX)  /reg2/ck  /reg3/ck
```

Gate /uu1 is a mux with an unknown driver, One of the pins, “d2”, looks like our clock, so the “sel” signal should have been “111”, but it is “XXX”. Trace /ten_reg.

```
report_gates /ten_reg

//  /ten_reg dff
//    ck    I  (XXX)  /in3
//    d     I  (111)  /gate5/out
//    q     O  (XXX)  /uu1sel
//    qbar  O  (XXX)
```

The gate /ten_reg is a dflop that controls the mux. In this design, this control signals is actually an internally generated “test_enable” signal. The flop needs to be initialized to hold a “111” on the output. Modify test_setup sequence as needed to put the device into a state so there is a sensitizable path from the scan chain output back to the scan chain input.

T4

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A memory element in the scan path must have an active clock during some time period of the **shift** procedure. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
Clock inputs of N (G) never set active during shift procedure. (T4-1)
```

N is the instance name and G is its gate ID number.

The summary message is:

```
T4: #fails=F handling=error (scan cell without active clock event during shift)
```

F is the number of T4 violations.

How to Debug T4 Violations in DFTVisualizer

You can debug T4 violations in the DFT visualizer. When the tool encounters the error, it will report a message:

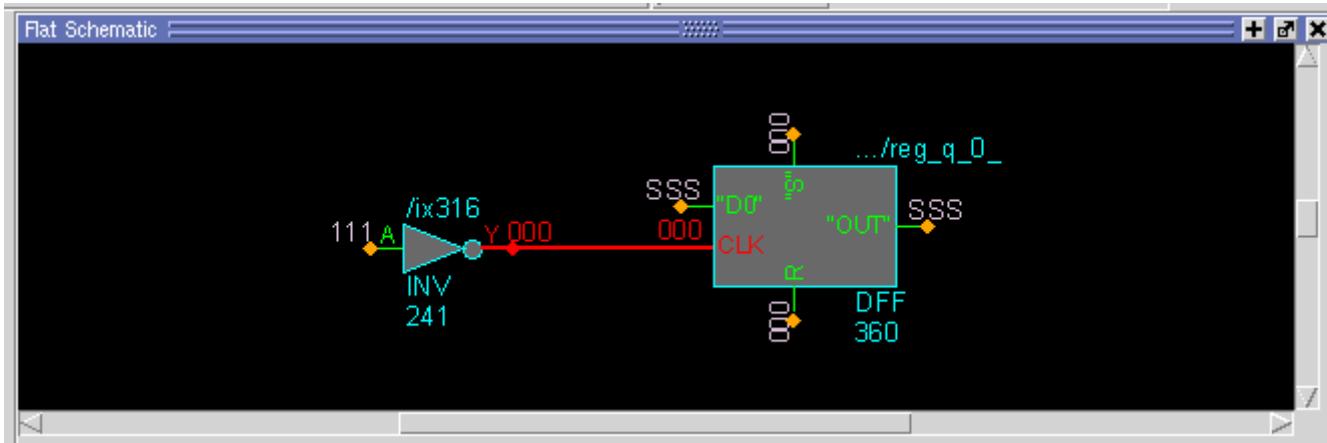
```
// Error: Clock inputs of /data0/reg_q_0_ (360) never set active during shift procedure. (T4-1)
```

Issue these commands:

```
set_gate_report trace
set_trace_report on
set_gate_level primitive
analyze_drcViolation T4-1
```

Since T4 and T3 DRCs occur in pairs, you can analyze T3-1. DFTVisualizer starts and displays the instance with the violation. [Figure 7-54](#) shows the signal driving CLK of /reg_q_0_ violates the T4 DRC. That clock must be active at some point.

Figure 7-54. T4 Violation



Trace back from the A input of the INV /ix316, and continue until you find the problem origin.

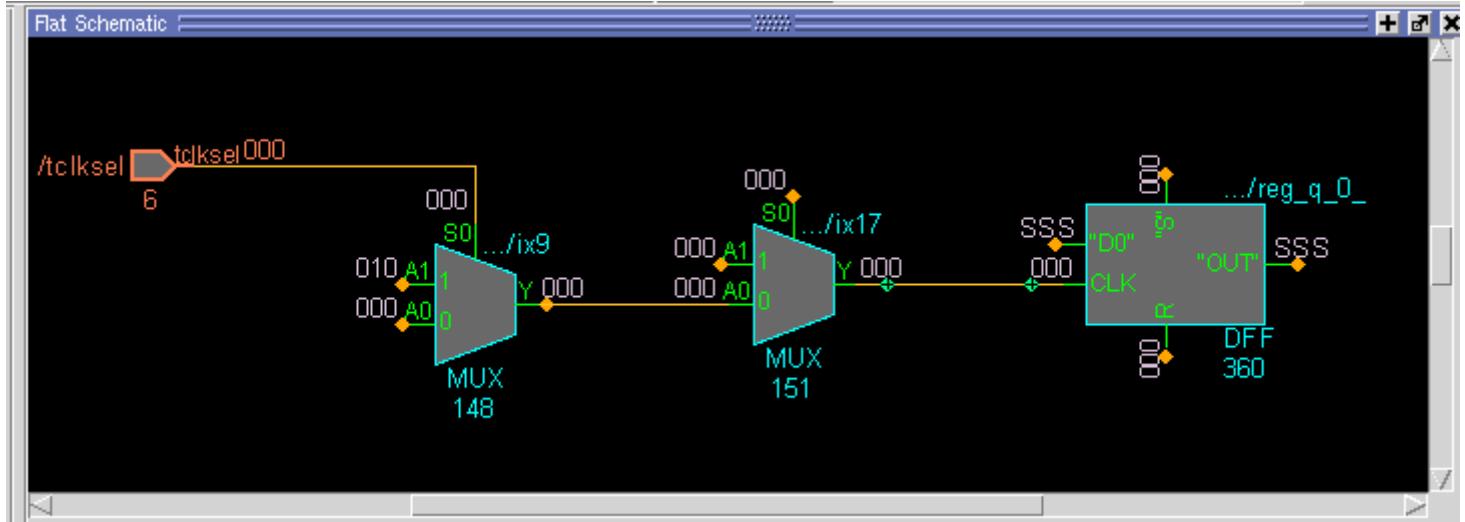
Note

When a T4 violation occurs, and you use the “analyze_drcViolation” command, the gate report is set to trace and the report will show an SSS on some paths. The S distinguishes scan path “don’t cares” from non-scan path “don’t cares”, which are shown as X.

The same happens when you have used the “set_gate_report drc shift” command.

Figure 7-55 shows the traceback. The control signal for a MUX, /ix17, is set to select input A0. It is driven by a MUX, /ix9, whose control signal, tclkSEL, is selecting A0 as well. However, the input A1 has the pulse that is required to fix the problem CLK. So we need to have tclkSEL go high.

Figure 7-55. T4 Violation Traced Back



You need to find when to pulse the offending clock high by forcing tclksel high at some point. You can determine when by setting the gate report to see the states of all the signals. This can be done with the command:

```
set_gate_report drc state
```

Figure 7-56 shows the offending /reg_q_0_ instance and the states of the signals. CLK is fine until the last shift, when it is 0.

Figure 7-56. T4 Violation Showing States

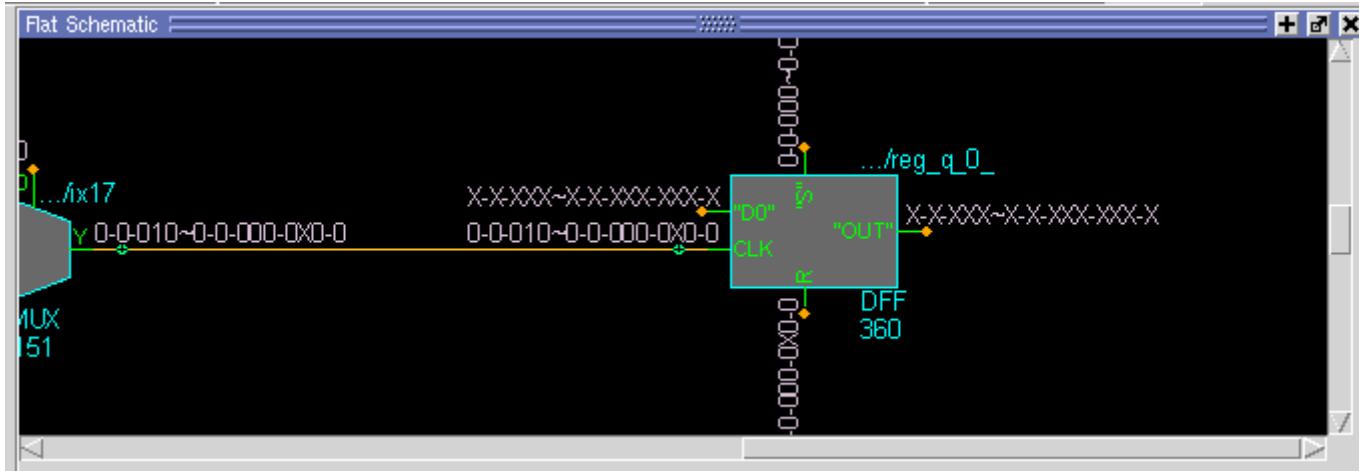
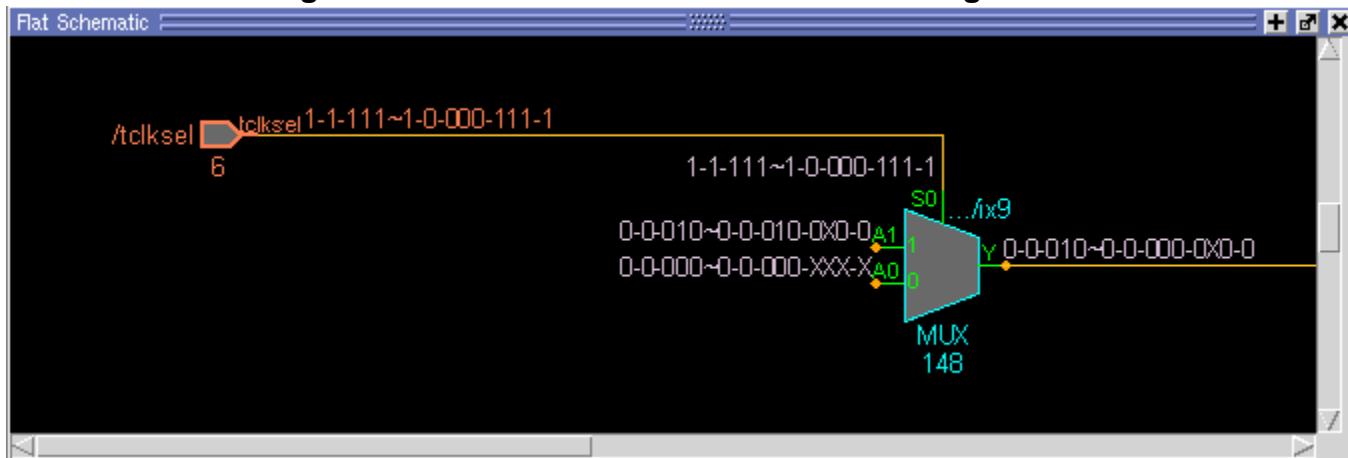


Figure 7-57 shows the result of tracing back to tclkSEL. The tclkSEL signal is 000 in the last shift and needs to be forced to 1 in the load_unload procedure.

Figure 7-57. T4 Violation Traced Back Showing States



The load_unload procedure in the test procedure file should look like this:

```
rocedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp2 ;
    cycle =
        force tclkSEL 1 ;
        force clear 0 ;
        force scan_en 1 ;
        force tclk 0 ;
        force clka 0 ;
        force clkB 0 ;
        force clkr 0 ;
        force clock 0 ;
    end ;
    apply shift 12;
    cycle =
        force tclkSEL 1 ;
    end ;
    apply shift 1;
end;
```

How to Debug T4 Violations Without DFTVisualizer

You correct T4 violations by making certain that there is an active clock during some time period of the shift procedure. Correct this error condition by accessing the simulated values of all the time periods of the shift procedure.

For simple cases set the gate reporting style to trace:

set_gate_report trace

Use the "report_gate" command to help you know where to trace back on the blocking path and then sensitize the path correctly.

For more complex cases set the gate report style to state:

set_gate_report drc state

Go back into analysis mode:

set_system_mode analysis

Use the “report_gate” command to help you know where to trace back on the blocking path.

Make the necessary changes to sensitize the path correctly.

There could be several causes for T4 violations, these are the most common issues

- The shift clock is defined incorrectly in the shift procedure
- A missing or incorrect pin constraint
- The TAP controller is in the wrong state

T5

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

During the **shift** procedure, you must never place an X value on a clock input or an active (X or 1) value on a set or reset input of a memory element in the scan path. The T5 DRC reports such issues.

Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

The error message is:

```
T input of N (G) set to V. (T5-1)
```

T is the type of input (clock, set, or reset), N is the instance name, G is its gate ID number, and V is the invalid state.

The summary message is:

```
T5: #fails=F handling=error (scan cell with unexpected clock event)
```

F is the number of fails for this DRC violation.

How to Debug T5 Violations in DFTVisualizer

You can debug T5 violations in the DFT visualizer. When the tool encounters the error, it will report a message:

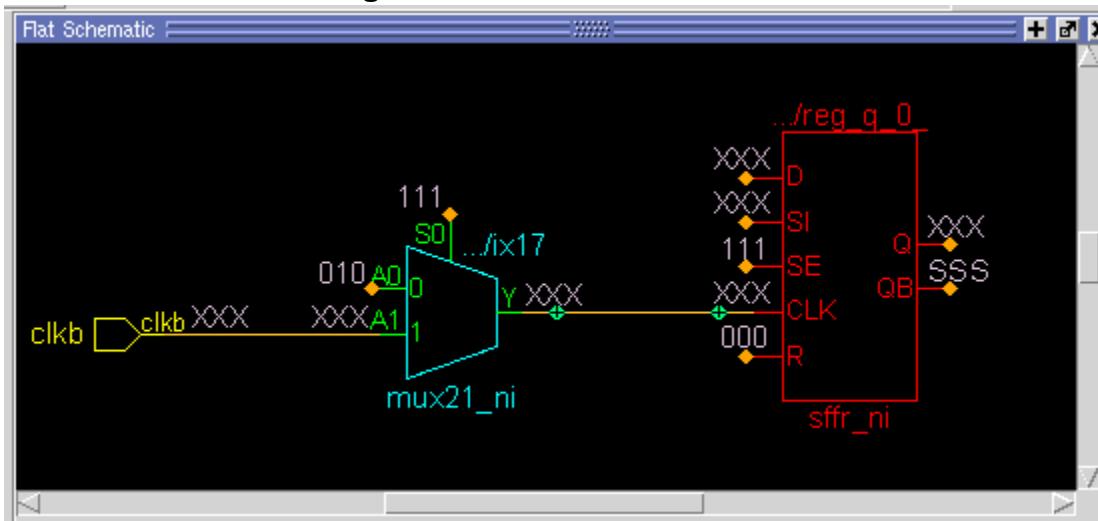
```
// Error: clock input of /data0/reg_q_0_ (360) set to X. (T5-1)
// Error: Scan chain chain1 blocked at gate /data0/reg_q_0_ (360) after
tracing 0 cells. (T3-1)
```

T5 and T3 DRC violations occur in pairs. Resolving one corrects the other. This debugging example begins with this command:

analyze_drcViolation T5-1

The DFTVisualizer opens displaying the instance with the T5 (and T3) violation as shown in [Figure 7-58](#)

Figure 7-58. T5 DRC Violation



The clock input, CLK, of the /datao/reg_q_0_ is set to XXX. That is the cause of both the T5 and T3 violations. The S0 pin of the MUX appears to be constrained to 1, which selects the A1 input of the MUX. The desired input is A0, which has a pulse 010.

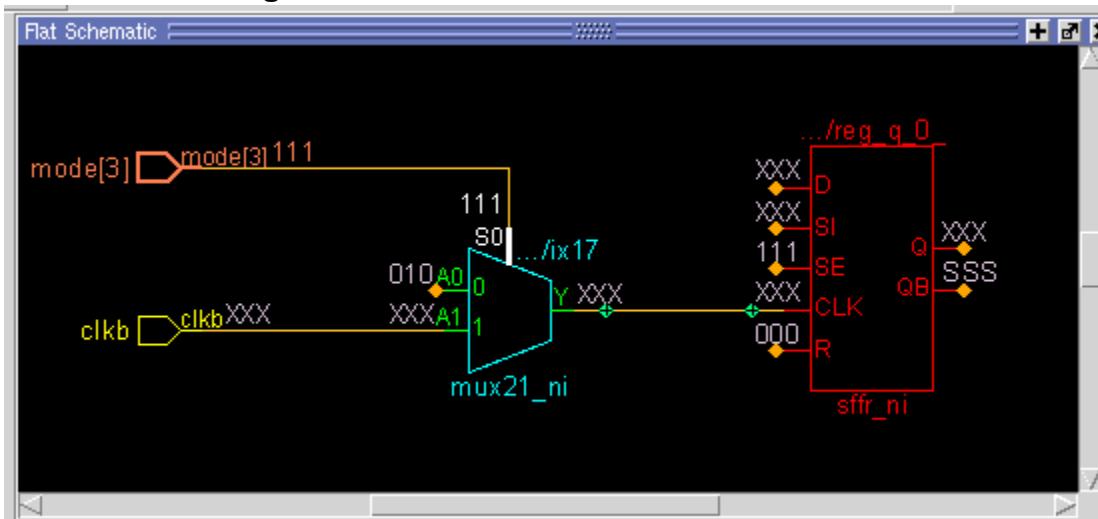
Note

When a T5 violation occurs, and you use the “analyze_drcViolation” command, the gate report is set to trace and the report will show an SSS on some paths. The S distinguishes scan path “don’t cares” from non-scan path “don’t cares”, which are shown as X.

The same happens when you have used the “set_gate_report drc shift” command.

Figure 7-59, shows S0 is driven by a primary input, mode[3].

Figure 7-59. T5 DRC Violation Traced Back



Use the command “report_input_constraints” to verify mode[3] is constrained. The solution for this would be to constrain the input S0 to 0.

This can be done temporarily in the visualizer with these commands:

```
delete_input_constraint mode[3]
add_input_constraint mode[3] -C0
set_system_mode analysis
```

You could also make certain the primary input is correctly constrained in a dofile.

How to Debug T5 DRC Violations Without DFTVisualizer

You correct T5 violations by making certain that during the shift procedure, you must never place an X value on a clock input or an active (X or 1) value on a set or reset input of a memory element in the scan path. Correct this error condition by accessing the simulated values of all the time periods of the shift procedure.

Set the gate reporting style to trace:

```
set_gate_report trace
```

Use the "report_gate" command to trace back on the blocking path to find the path that corrected.

T6

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

During any time period of the **shift** procedure, a memory element in the scan path must never have more than one clock input turned on. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

The error message is:

```
Multiple clock inputs of N (G) set active. (T6-1)
```

N is the instance name and G is the gate ID number.

T8

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The **measure_sco** statement in the **shift** procedure must follow the successful observation of the scan cells.

To guarantee the observation is successful, the time of the **measure_sco** statement must meet the following conditions:

- It must not occur after exercising the first clock on the memory element closest to the scan chain output.
- It cannot be at time 0 if you capture the data into the last memory element at the end of the **shift** procedure.
- The states on all inputs at the **measure_sco** time must be the same as at the end of the **shift** procedure.

To correct the error condition, you must modify the **shift** procedure to meet the above conditions for the **measure_sco** statement.

The error message is:

```
Invalid measure_sco time. (T8-1)
```

T9

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The traced scan chain input pin must be the same as the scan chain input pin specified with the add_scan_chains command. You can correct this error condition by redefining the scan chain input to be the traced pin.

The error message is:

```
Chain input P1 doesn't match entered value P2. (T9-1)
```

P1 is the name of the traced scan chain input pin, and P2 is the name of the entered scan chain input pin.

T10

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The time of the **force_sci** statement in the **shift** procedure must occur before a clock input of the memory element (closest to the scan chain input) turns on. Correct this error condition by changing the time of the **force_sci** statement to a value less than or equal to the indicated time.

The error message is:

```
Force_sci must occur on or before time T. (T10-1)
```

T is the maximum time.

T11

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A clock input of the memory element (closest to the scan chain input) must not turn on during the **shift** procedure prior to the time of the **force_sci** statement. Correct this error condition by changing the times of the **force_sci** statement or **force** statements.

The error message is:

```
Incorrect propagation of force_sci value to scan cell. (T11-1)
```

T12

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

If a scan cell contains a SLAVE element, the MASTER element is not directly observable.

The error message is:

```
MASTER not observable, a master_observe procedure is required by scan group G. (T12-1)
```

G is the scan group.

When the handling is set to other than error, the tool automatically makes the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERS, test coverage may be reduced. To correct this violation, you can define a master_observe procedure to propagate the MASTER value to the SLAVE.

T13

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you define and try to use a **master_observe** procedure, there must be at least one scan cell that contains a SLAVE. If there is no such cell, Correct this error condition by deleting the **master_observe** procedure.

The error message is:

```
Master_observe procedure defined but not used. (T13-1)
```

T14

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you define and try to use a **shadow_control** procedure, there must be at least one identified SHADOW memory element. Correct this error condition by either changing or deleting the **shadow_control** procedure.

The error message is:

```
No SHADOWs identified using shadow_control procedure. (T14-1)
```

T15

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define and try to use a **shadow_observe** procedure, the procedure must observe at least one SHADOW. You can correct this error condition by changing or deleting the **shadow_observe** procedure.

The error message is:

```
No observable SHADOWs identified using shadow_observe procedure. (T15-1)
```

T16

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink gate (PGS) in the long path must be at the non-controlling value of the PGS gate.

To correct this error condition, access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. You can also avoid this error by setting the pulse generators to off, but that results in no pulse generator support.

The error message is:

```
Input of pulse generator N (G) not at correct value when clocks are off.  
(T16-1)
```

N is the pulse generator instance name, and G is its gate ID number.

T17

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Reconvergent pulse generator sink gates (PGS) cannot connect to any of the following: primary outputs, non-clock inputs of scan memory elements, ROM gates, non-write inputs of RAMs, or transparent latches. To avoid this error, set the pulse generators to off, however note that this results in no pulse generator support.

The error message is:

```
Pulse generator N1 (G1) connected to T N2. (T17-1)
```

N1 is the pulse generator instance name, and G1 is its gate ID number.

T18

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported

The maximum number of traced cells in the longest scan chain of a group must equal the entered number of repetitions in the **apply shift** statement in the **load_unload** procedure. The default setting for this DRC is “Ignore.”

You can enable this check with the [set_drc_handling](#) command.

You can correct this condition by changing the repetition number on the **apply shift** statement. This rules violation has no adverse effects because the tool recalculates the actual number of necessary shifts based on the number of scan cells it encounters.

The warning message is:

```
Traced number of shifts (N1) doesn't match entered value (N2). (T18-1)
```

N1 is the traced number of shifts, and N2 is the entered number of shifts.

T19

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

If one scan cell has a SLAVE, then all scan cells must have a SLAVE. You must correct this warning by changing the netlist of the scan chains.

The warning message is:

```
N scan cells do not have a SLAVE when some do. (T19-1)
```

N is the number of non-slave scan cells.

T20

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The number of shifts specified using the set_number_shifts command must be at least equal to the length of the longest scan chain. To correct this error set a valid value using the set_number_shifts command.

The error message is:

Entered number of shifts N is too small. (T20-1)

N is the entered number of shifts, and T20 is the rule ID number.

T21

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The number of independent **shift** applications in the **load_unload** procedure must be less than the scan chain length. Correct this error by removing a sufficient number of independent **shift** applications from the **load_unload** procedure or by deleting the short scan chain.

The error message is:

Number of independent shifts N must be less than scan chain length L.
(T21-1)

N is the number of independent **shift** applications, L is the scan chain length, and T21 is the rule ID number.

T22

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If the rules checker traces a scan cell during the application of an independent **shift**, it must also trace that cell during the application of its associated general **shift**. Correct this error by

changing the sensitization for either the independent or general **shift**, so that they are sensitizing the same scan cells.

The error message is:

```
N (G) was not used in general chain trace. (T22-1)
```

N is the scan cell instance name, G is its gate ID number, and T22 is the rule ID number.

T23

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The chain length calculated for an independent **shift** must be the same as that calculated for its associated general **shift**. Correct this error by changing the sensitization for either the independent or general **shift**, so that they are sensitizing the same scan cells.

The error message is:

```
Chain length (L1) using independent shift not equal to chain length (L2).  
(T23-1)
```

L1 is the independent shift chain length, L2 is the general shift chain length, and T23 is the rule ID number.

T24

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When two adjacent memory elements (source and sink) in a scan chain path are clocked by different shift clocks, the sink must not capture data from the source at the same time the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan chain shifting when clock skew exists between the different shift clocks.

The warning message is:

```
A lockup latch may be required between N1 (G1) of cell M1 and N2 (G2) of  
cell M2 in scan chain S. (T24-1)
```

N1 and N2 are the instance names of the adjacent elements clocked by different clocks. G1 and G2 are the corresponding gate IDs. M1(M2) is the cell ID of the scan cell in which N1(N2) is located. S is the name of the scan chain.

Note

If N1(N2) is not within a scan cell, M1(M2) is the cell ID of the scan cell that is driven by N1(N2).

You can correct the violation by inserting a lockup latch between the adjacent memory elements reported in the violation message. To find out which clocks are involved, use the command, [report_scan_cells -Range cell_id1 cell_id2](#).

T25

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

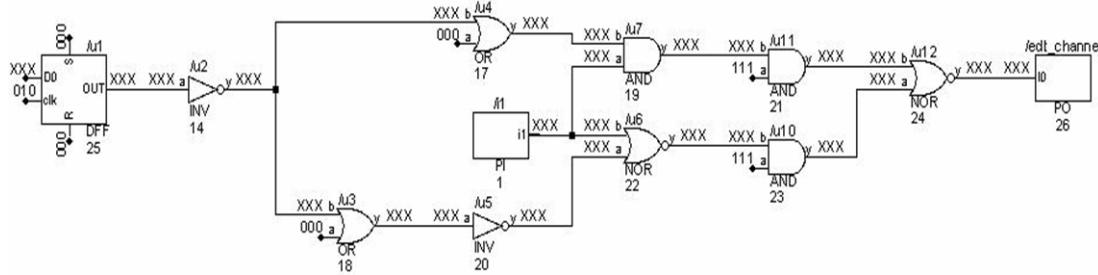
Default Handling: Warning

report_drc_rules: Supported

Scan chain tracing must be independent of whether or not a gate with an X value is initialized to a known value (either 0 or 1).

[Figure 7-60](#) is an example of a T25 violation.

Figure 7-60. Example of T25 Violation



In [Figure 7-60](#), the scan path is from DFF gate 25 to PO gate 26. If PI gate 1 is initialized to 1, then the actual scan path is the upper path from gate 25 to gate 26. If PI gate 1 is initialized to 0, then the actual scan path is the lower path from gate 25 to gate 26. In either case, the scan shifting is successful; however, a simulation mismatch could occur in Verilog if PI gate 1 is not initialized (remains an X value). The warning message is:

There is a T25 violation. Scan chain tracing has assumed that some gates with X values are initialized to any non-X values. This is not a problem in silicon but may cause Verilog simulation mismatches. For more conservative scan chain tracing, please use the command: "set_drc_handling -scan_chain_tracing conservative".

You can display the T25 violation using the “report drc rule T25” command as shown in the following example:

rep drc rule T25

```
Warning: Scan chain tracing between scan output and scan cell 0
(from gate 24 back to gate 14) in scan chain 0 assumes that some gates
with X values are initialized to any non-X values. (T25-1)
```

You can debug T25 violations by displaying the gate value in scan shift. To do this, use the [set_gate_report](#) command to set gate reporting to drc shift and use the [report_gates](#) command with the gate ID number displayed in the message.

The default handling for this rule violation is warning. If you do not want scan chain tracing to pass with a T25 violation, use the “set_drc_handling -scan_chain_tracing conservative” command. For more information, see the [set_drc_handling](#) command.

T26

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The shift procedure must create a sensitizable path through bidirectional pins at the scan chain output or EDT channel output. The improperly sensitized bidirectional pin causes an error condition.

For scan chain output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional scan chain output pin N in
chain "C". The pin is not forced to Z during measure_sco. (T26-1)
```

For EDT channel output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional channel output pin N in block
"B". The pin is not forced to Z during measure_sco. (T26-2)
```

N is the pin name, C is the corresponding chain name, and B is the corresponding EDT block name.

You can avoid this rule violation by adding a force to Z for the reported pins at the beginning of the load_unload procedure. Alternatively, you can force all bidirectional pins with a ‘force _ALL_BIDI Z’ at the beginning of the load_unload procedure. The _ALL_BIDI is a macro that selects all primary bidirectional pins.

You can also define the pin as an output pin instead of as an inout pin in your Verilog file. By doing this, then there is no need to force this pin to Z anymore.

Power-Aware Rules (V Rules)

Power-aware designs rely on DRC rules that are designed specifically for that application.

You must load power data using the `read_cpf` or `read_upf` commands before using the `report_drc_rules` command for the V DRC rules.

When using the power-aware DRC rules, the following conditions apply:

- **Rule V1 to V7** — Checked after reading a power data file.
- **Rule V8 to V21** — Checked when switching from setup mode to a non-Setup mode.

The tool displays a summary of the V rules when you load power data into the tool without any parsing errors as shown here:

```
read_upf my_design.upf
report_drc_rules -summary

...
T26: #fails=0 handling=error (bidirectional chain / channel outputs not
forced to Z during chain tracing)
V1: #fails=0 handling=ignore/verbose (no power modes are defined)
V2: #fails=0 handling=ignore/verbose (no default power modes are defined)
V3: #fails=0 handling=error (multiple default power modes are defined)
V4: #fails=0 handling=error (no default power domains are defined)
V5: #fails=0 handling=error (multiple default power domains are defined)
V6: #fails=0 handling=warning (no restore_edge or default_restore_edge is
defined)
V7: #fails=0 handling=warning (power mode is not reachable)
V8: #fails=0 handling=error (scan data path is in a power-off domain)
V9: #fails=0 handling=error (scan control path is in a power-off domain)
V10: #fails=0 handling=error (power start/end/shutoff condition is X
during the scan shift period)
V11: #fails=0 handling=error (active power mode has changed during the
scan shift period)
V12: #fails=0 handling=warning (active power mode may change during the
capture cycle)
V13: #fails=0 handling=warning (path crosses power domains without a
level-shifter cell)
V14: #fails=0 handling=warning (path crosses power domains without an
isolation cell)
V15: #fails=0 handling=note (scan chain contains both retention and
regular cells)
V16: #fails=0 handling=note (scan chain contains both always-on and
other types of cells)
V17: #fails=0 handling=note (EDT channel contains both retention and
regular cells)
V18: #fails=0 handling=warning (power control signals are in a domain
that could be powered-off)
V19: #fails=0 handling=warning (retention save signal is not off during
shift)
V20: #fails=0 handling=note (non-scan retention cells are identified in
the design)
V21: #fails=0 handling=error (power data assertions are detected)
```

V1.....	3019
V2.....	3020
V3.....	3020
V4.....	3020
V5.....	3021
V6.....	3021
V7.....	3022
V8.....	3022
V9.....	3023
V10.....	3024
V11.....	3025
V12.....	3025
V13.....	3026
V14.....	3027
V15.....	3028
V16.....	3028
V17.....	3028
V18.....	3029
V19.....	3029
V20.....	3030
V21.....	3030

V1

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No power modes are defined when there are multiple power domains. You cannot change the handling.

The violation message is:

At least one power mode is required when multiple power domains are defined (V1).

Because power modes in UPF are implicitly learned through power switches control logic, the V1 rule is ignored in UPF format.

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V2

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No default power modes are defined when there are multiple power modes. You cannot change the handling.

The violation message is:

```
A default power mode is required when multiple power modes are defined  
(V2) .
```

The V2 rule is ignored for UPF format.

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V3

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Multiple default power modes are defined. You cannot change the handling.

The violation message is:

```
Multiple default power modes are defined (V3) .
```

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line containing the error.

V4

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No default power domain is defined when there are multiple power domains. You cannot change the handling.

The violation message is:

A default power domain is required when multiple power domains are defined (V4) .

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V5

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Multiple default power domains are defined. You cannot change the handling.

The violation message is:

Multiple default power domains are defined (V5) .

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line containing the error.

V6

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Either restore_edge or default_restore_edge must be defined if the design contains any retention cells; if this rule is violated, the state retention rule is ignored and no retention value is restored.

Note that missing both save_edge and default_save_edge is allowed on CPF as long as restore_edge or default_restore_edge is defined. In this case, the inversion of restore_edge (or default_restore_edge when restore_edge is missing) will be used as save_edge.

The violation message is:

No restore_edge is defined for the state retention rule name. (V6-n) .

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the retention definition lines.

V7

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A defined power mode is not reachable from the default power mode with the defined power transition rules.

The violation message is:

```
Power mode name1 is not reachable from the default power mode name2 using  
any defined power transition rule (V7-n).
```

The V7 rule is ignored for UPF format.

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line defining the power mode.

V8

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The scan tracing should not propagate to a scan cell in a power domain that is in the shutoff state.

Note

 Scan insertion in the dft -scan contexts supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

Note that the number of power OFF gates in a given scan chain can be large, so only one V8 violation is recorded per scan chain and the first scan path gate from the sco pin is reported. Violation of the V8 rule can be due to at least one of the following reasons:

- The power mode definition is incorrect.
- The shutoff condition of the power domain is defined incorrectly.

- The test procedures (for example, load_unload or shift) for the scan chain operation are defined incorrectly.
- The inserted scan logic of the design does not consider the power mode properly.

The violation message is:

The scan path of chain chain-name is in power off domain domain-name (the first power off scan path gate is N1 (G1) after tracing C cells) (V8-n)

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer displays the scan cell in the Flat Schematic window with a callout box attached containing the following message:

Power Domain: < name>, Power Mode (CPF) or Power State: <name>

V9

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

All control logic for the scan operation must be in a power ON domain during the entire scan shift operation.

Note

 Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

Some popular control logic examples are listed below:

- If EDT logic exists in the design, it should stay at power ON mode during the scan test when the compression mode is used; otherwise the scan operation will fail.
- If the scan_enable signal is derived from a JTAG controller, the JTAG controller should stay at power ON mode during the scan test.
- The shift clock control logic should remain at power ON mode during the scan test. The tool performs the check by monitoring all test procedures except test_setup and ensures that the power domains where the scan control logic are located remain at the power ON mode after test_setup procedure.

The violation message is:

The scan chain chain-name is blocked at N1 (G1) after tracing C cells due to the control gate N2 (G2) in power off domain domain_name (V9-n).

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer displays the control gates in the Flat Schematic window with a callout box attached containing the following message:

```
Power Domain: < name>, Power Mode (CPF) or Power State: <name>
```

V10

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The expression used by mode_transition rule and shutoff condition should not be X during the scan shift period. The tool performs the rule by checking all events to ensure that no expression can be X during the scan shift operation. The expressions to be checked include the shutoff condition of every power domain and the start and end conditions of every mode_transition rule.

Note

 Scan insertion in the dft -scan contexts supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

The violation messages are:

Case 1 — When shutoff condition of a power domain is X:

```
The shutoff condition of power domain name is X at time t of procedure
procedure-name (V10-n).
```

Case 2 — When the start condition of a mode_transition rule is X:

```
The start condition of mode transition rule name is X at time t of
procedure procedure-name (V10-n).
```

Case 3 — When the end condition of a mode_transition rule is X:

```
The end condition of mode transition rule name is X at time t of procedure
procedure-name (V10-n).
```

The handling of X is different based on the handling of the V10 DRC rule. When the handling of the rule is set to warning, the condition of X will be treated as HI as if the condition is satisfied. When the handling is Note, the condition of X will be treated as LO as if the condition is not satisfied. Note, the violation of this rule may cause the design to be in the wrong power mode and thereby cause a simulation mismatch.

When you issue the analyze_drcViolation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V11

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

During scan chain shift operation, the design should not change power modes because this can cause some power domains to be switched between power OFF mode and power ON mode. This occurs because changing active power domains during the shift period may cause the scan chain configuration to be changed. The tool performs the rule by checking the shutoff expressions of all power domains to ensure they remain false during the scan shift period.

Note

 Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

The violation messages are:

Case 1 — When mode transition rules are defined:

```
Active power mode changed from mode1 to mode2 at time t of procedure
procedure_name (V11-n).
```

Case 2 — When mode transition rules are not defined:

```
The power domain name changed from state-1 to state-2 at time t of
procedure procedure_name (V11-n).
```

When the handling of the violation is not error, the scan shift operation may be unreliable and this can result in a simulation mismatch. Note that a mismatch may not be seen in the parallel test bench and this can make the debugging process more complicated.

When you issue the analyze_drcViolation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V12

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

ATPG should not create_patterns that can turn off an active power domain during the capture cycle; turning off an active power domain during the capture period changes the scan chain configuration. The tool performs the rule by checking on active power domains with shutoff expressions and ensuring they cannot be true during the capture period.

Note

 Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

The violation messages are:

Case 1 — When mode transition rules are defined:

The power mode may change from name1 to name1 during the capture cycle (V12-n) .

Case 2 — When mode transition rules are not defined:

The power domain name may change state during the capture cycle (V12-n) .

When the shutoff condition is true and the handling of the violation is not error, the capture value of the scan cells in the shutoff power domain remains the same as if the power domain is still active. This may cause some test patterns to fail in silicon that may not be detected during the logic simulation phase.

When you issue the analyze_drcViolation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V13

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A path should not cross different power domains without a level-shifter cell if the power domains can operate in different operation voltages.

The following situations can cause a V13 DRC violation:

- An instance is driven by another instance of a different voltage power domain.

- The scan_in pin or the EDT decompressor to the first scan cell crosses different voltage domains.
- The last scan cell to the scan_out pin or EDT compactor crosses different voltage domains.
- The scan cell (i+1) to scan cell i crosses different voltage domains.

The violation messages are:

Case 1 — When violation is on a non-scan path:

Path from gate N1 (G1) to gate N2 (G2) crosses different voltage domains without a level-shifter cell (V13-n).

Case 2 — When violation is on a scan path:

Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan chain chain-name crosses different voltage domains without a level-shifter cell (V13-n).

When you issue the analyze_drcViolation command on an occurrence of this violation, DFTVisualizer displays the path between the two gates reported in the message in the Flat Schematic window and displays the simulation data as a DRC in the Flat Schematic window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

V14

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

If a path crosses two power domains and the driving power domain contains the shutoff condition, the path must include an isolation cell. Similar to rule V13, there are four situations to be considered for this rule.

The violation messages are:

Case 1 — When violation is on a non-scan path:

Path from gate N1 (G1) to gate N2 (G2) crosses different power domains without an isolation cell (V14-n).

Case 2 — When violation is on a scan path:

Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan chain chain-name crosses different power domains without an isolation cell (V14-n).

When you issue the analyze_drcViolation command on an occurrence of this violation, DFTVisualizer displays the path between the two gates reported in the message in the Flat Schematic window and displays the simulation data as a DRC in the Flat Schematic window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

V15

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

A scan chain should not include both retention cells and regular scan cells. A violation of this rule will make retention test less efficient.

The violation message is:

```
Scan chain chain-name contains both retention cells and regular cells  
(V15-n) .
```

This violation is not supported in DFTVisualizer.

V16

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Typically, always-on cells are used to program power mode and retention mode. To make the control of the power mode more efficient, a scan chain should not include both always-on cells and other types of scan cells.

The violation message is:

```
Scan chain chain-name contains both always-on cells and other type of  
cells (V16-n) .
```

This violation is not supported in DFTVisualizer.

V17

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

An EDT scan channel should try to prevent the mix of scan chain with retention cells and, regular scan chain without retention cells. This rule prevents the retention cells from being masked by regular cells when the circuit is restored from the retention mode.

The violation message is:

```
Scan channel channel-name contains both retention scan chain and regular  
scan chain (V17-n) .
```

This violation is not supported in DFTVisualizer.

V18

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All power control signals, including retention save, retention restore, isolation_enable, and power-domain shutoff conditions should be from always-on power domains that cannot be turned off.

By default, the V18 violation handling is Warning. The tool starts with an all power-off state, except for the always-on power domains.

The violation message is:

```
Shutoff condition 'condition_expression' for domain domain_name is in non-  
always-on power domain (V18-n) .
```

When the V18 violation handling is set to Note, the tool assumes it is in an all power-on state at the beginning of test_setup.

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file and highlights the line that defines the power domain.

V19

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Retention save signals should be off during the scan shift period to prevent the loading value from being overridden by the retention saved state.

The violation message is:

```
Save signal 'condition_expression' for retention rule rule_name is not off  
during the scan shift period (V19-n) .
```

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file and highlights the line that defines the rule name.

V20

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Only scanned retention cells are supported by the retention test to test the retention functionality. So this rule checks if there are any non-scan retention cells.

The violation message is:

```
Retention cell cell_name is a non-scan cell (V20-n) .
```

This violation is not supported in DFTVisualizer.

V21

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The design should never enter any illegal power configurations specified in CPF command assert_illegal_domain_configurations during any test cycle.

The violation message is:

```
The active power mode violates assert_illegal_domain_configurations name  
at time t of procedure proc_name (V21-n) .
```

Note

 CPF and UPF power data parsers support the Tcl scripting mechanism. The violation of the Tcl format results in an error. Note, the format of the error message is not described in this document.

When you issue the [analyze_drcViolation](#) command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. In the CPF file, DFTVisualizer highlights the assert_illegal_domain_configurations command (if applicable).

Timing Rules (W Rules)

These rules apply to timeplates and the mapping of timeplates to procedures. This includes making sure that the order of events in a procedure is not changed after leaving setup mode.

For timing rules specific to the enhanced procedure file, see “[Test Procedure File](#)” in the *Tessent Shell User’s Manual*.

W1	3033
W2	3033
W3	3034
W4	3034
W5	3034
W6	3035
W7	3035
W8	3036
W9	3036
W10	3036
W11	3037
W12	3037
W13	3038
W14	3038
W15	3038
W17	3039
W18	3039
W19	3040
W20	3040
W21	3041
W22	3041
W23	3041
W24	3042
W25	3042
W26	3043
W27	3043
W28	3044
W29	3044
W30	3045

W31	3045
W32	3046
W33	3046
W34	3047
W35	3048
W36	3048
W37	3048
W38	3049
W39	3050
W41	3051

W1

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This message can only occur when not in setup mode. After leaving setup mode, it is possible to load an enhanced procedure file using the read_procfile command. It is also possible for the enhanced procedure file to load a new procedure which overwrites a procedure already loaded. This is fine as long as the event order in the new procedure matches that of the old procedure.

If the new procedure has additional events, the tool will issue this error message:

```
New procedure P has more events than existing procedure. (W1)
```

P is the name of the new procedure.

W2

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Similar to rule W1, but this rule is violated if the new procedure has less events than the old procedure.

The error message is:

```
New procedure P has fewer events than existing procedure. (W2)
```

P is the name of the new procedure.

W3

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Similar to rule W1, this rule is violated if the new procedure has a different event order than the old procedure.

The error message is:

```
New procedure P has a different event order than existing procedure. (W3)
```

P is the name of the new procedure.

W4

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This message is similar to rule W1. Instead of an entire new procedure being loaded, the enhanced procedure file only specifies that a new timeplate is applied to an existing procedure. This is done by specifying a new procedure in the enhanced procedure file with only a timeplate reference statement.

If that new timeplate causes the event order in the old procedure to change, the tool will issue this error message:

```
New timeplate T changes event order in procedure. (W4)
```

T is the name of the new timeplate.

W5

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A new non-scan procedure has an event statement that either occurs in the wrong order, or is not allowed in that type of procedure.

All non-scan procedures must conform to the event order that is stated for them in “Non-Scan Procedures” in the *Tessent Shell User’s Manual*. An example of this would be a capture procedure that has a “measure_po” statement before the “force_pi” statement.

The error message is:

```
Procedure P has an illegal event statement or event order S. (W5)
```

P is the procedure name and S is the illegal statement or event order.

W6

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If one shift procedure has the “measure_sco” statement occurring after the pulse of the shift clock, then all shift procedures must place the “measure_sco” statement after the pulse of the shift clock, and all load_unload procedures must have a “measure_sco” statement occurring at the end of the cycle right before the “apply shift” statement. Placing the “measure_sco” statement after the pulse statement enables end measure mode which affects the way the Vector Interfaces code writes out parallel load scan patterns.

See the “[Creating Test Procedure Files for End Measure Mode](#)” section of the *Tessent Shell User’s Manual*.

The error message is:

```
Shift procedure P [does | does not] use end measure, while previous ones [do | do not]. (W6)
```

P is the procedure name.

W7

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

No timing information has been specified for the named procedure. Either the procedure needs to reference a timeplate, or time values must be associated with the event statements.

The error message is:

No timeplate or times specified for procedure P. (W7)

P is the procedure name.

W8

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A scan procedure (shift, load_unload, ...) has been read in by the read_procfile command, but no “scan_group” statement is in the procedure.

The error message is:

No scan group specified for procedure P. (W8)

P is the procedure name.

W9

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named procedure has no event statements, no apply statements, and no timeplate references.

The error message is:

No events in procedure P. (W9)

P is the procedure name.

W10

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This warning is issued during DRC checking. It indicates that none of the procedure files specified in the add_scan_groups commands have any procedures in them.

The warning message is:

```
No test procedures have been loaded. (W10)
```

W11

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A cycle in a procedure that needs to be split into two cycles in order to use the timeplate that is specified for that procedure. This could occur, for example, if the test_setup procedure contains a pulse statement on a specific clock pin, and yet the timeplate used referenced by this procedure does not contain a pulse statement for the clock. The user will receive a P53 rule message and then a W11 rule message to indicate that the cycle (where the pulse statement is) was split into two cycles in order to create the clock pulse.

The warning message is:

```
Cycle S being split in procedure P. (W11)
```

S in the cycle and P is the procedure name.

W12

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A timeplate was loaded that is missing a required statement. For example, a timeplate must have a “force_pi” statement. If a required clock pulse statement is missing from the loaded timeplate, you will receive a P53 rule message.

The error message is:

```
No S statement in timeplate T. (W12)
```

S is the statement name and T is the timeplate.

W13

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The event order for the name procedure has changed while parsing an procedure file, but since the tool is still in setup mode, this change of event order is acceptable. This could happen if two procedure files in two different add_scan_groups commands both have the same procedure (test_setup, for example), but one specifies a different event order than the other. The last one loaded is what will be used.

The warning message is:

```
Event order has changed in procedure P. (W13)
```

P is the procedure name.

W14

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Not Supported

This message indicates that a procedure has been replaced by a new procedure with the same name, same type, and same scan group if applicable. This message indicates that the replacement did not violate any rules. Both procedures have the exact same event order, but they might use different timeplates.

The warning message is:

```
Procedure P replaces same procedure from file S. (W14)
```

P is the procedure name and S is the filename.

W15

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

End measure mode is enabled but an illegal force or pulse statement occurs between the measure_sco statement and the end of the procedure in a shift procedure, or the illegal statement

occurs between the measure_sco statement and the apply shift statement in a load_unload procedure.

See the “[Creating Test Procedure Files for End Measure Mode](#)” section of the *Tessent Shell User’s Manual*.

The error message is:

```
Statement S cannot follow measure_sco in end measure procedure P. (W15)
```

S is the illegal statement and P is the procedure name.

W17

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Clock_po procedures are used to hold clock pins at certain values. In order to obtain the correct fault coverage, it is important that the timeplates used in a clock_po procedure do not pulse the clock pins, but only force them. If this is not corrected before patterns are saved, the Vector Interfaces code places X values on the output pins for the clock_po procedure and fault coverage decreases.

The error message is:

```
Timeplate T has clock pulses but is used in a clock_po procedure. (W17)
```

T specifies which timeplate has the clock pulses. To correct this, make sure you define a different timeplate that does not have clock pulses and use this in the clock_po procedure.

W18

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (can be changed to Note or Ignore)

report_drc_rules: Supported

Many ASIC vendors require that shift procedures only be one cycle in length, and it is very unusual to encounter a multiple cycle shift procedure. This warning alerts the user that something might not be specified correctly in the shift procedure, such as forcing the same pin with more than one value, or pulsing a clock twice.

The warning message is:

Shift procedure has more than one cycle, please check if shift clocks are defined properly. (W18)

W19

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This message occurs when using the offstate option improperly. The offstate option is only supported when using the Enhanced Vector Interfaces. Also, the pin referenced with the offstate option cannot be used in a capture, ram_sequential, or clock_sequential procedure. These are considered non-scan procedures.

Any timeplate that uses the offstate statement will be treated as a complex timeplate and will not be allowed to be used in a non-scan procedure. Complex timeplates are most useful in shift procedures where a non-clock pin needs to be pulsed while still maintaining a single cycle in the shift procedure. If you attempt to pulse a signal specified with an offstate option in a capture or other non-scan procedure, you will encounter the following error message.

Timeplate gen_tp1 with complex waveforms cannot be used in non-scan procedure capture. (W19)

W20

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This error occurs if, in a capture procedure, a force_pi event does not occur before the first clock pulse.

The error message is:

A force_pi statement does not exist before the first clock pulse in Procedure P. (W20)

P is the procedure name.

To correct this violation, add a force_pi statement to the beginning cycle of the procedure, before the first clock pulse. For a named capture procedure with internal and external modes, add a force_pi statement to the beginning cycle in each mode.

W21

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In a named capture procedure, a measure_po statement is allowed only in the last cycle of the internal mode. This error occurs if a measure_po statement is found in a cycle where it is not permitted.

The error message is:

Procedure P has one or more measure_po statements which do not occur in the cycle before the last clock pulse. (W21)

P is the procedure name.

To correct this violation, remove the measure_po statements from the cycles where they are not allowed.

W22

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, a measure_po statement is allowed only in the last “cyclized” ATPG cycle of the internal mode—before the last clock pulse. This warning alerts you that the tool did not find a measure_po statement in this location. A capture procedure with this warning will not use any PO as a capture point.

The warning message is:

Procedure P has no measure_po statement in the cycle before the last clock pulse. (W22)

P is the procedure name.

To view the cyclized information for the procedure, use the report_capture_procedures command.

W23

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This warning alerts you that the external mode's period is longer than the internal mode's period.

The warning message is:

The total period for the external mode is longer than the total period for internal mode in procedure P. (W23)

P is the procedure name.

To eliminate this warning, modify the timeplate(s) so that the external mode period matches the internal mode period.

W24

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This error occurs if the internal mode's period is longer than the external mode's period.

The error message is:

The total period for the internal mode is longer than the total period for external mode in procedure P. (W24)

P is the procedure name.

To correct this violation, modify the timeplate(s) so that the external mode period matches the internal mode period.

W25

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Event times in the external and internal modes of a named capture procedure should match. This warning alerts you that an event in the external mode does not have a matching event in the internal mode.

The warning message is:

```
// Warning: External event pulse S at time T needs to have a matching
internal event in procedure P or the signal needs to be disconnected from
internal sequential elements or observe points. (W25-1)
```

Where S is the signal name, T is the time of the event, and P is the procedure name.

In Tessent Diagnosis, these violations if uncorrected can cause pattern verification errors in the tool. See the *Tessent Diagnosis User's Manual* for more information.

W26

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Event times in the external and internal modes of a named capture procedure must match. This error occurs if an event (other than an internal pin event) in the internal mode does not have a matching event in the external mode.

The error message is:

```
Event E N at time T in internal mode does not have matching event in
external mode in procedure P. (W26)
```

E is the type of event, N is the name of the pin to which the event applies, T is the time of the event, and P is the procedure name.

To correct this violation, you need to do two things:

- Add the missing event to the external mode.
- Modify the timeplate(s) so that the event added to the external mode occurs at the same time as in the internal mode.

W27

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

In a named capture procedure, the start time for the scan load cycle in the internal mode must have a match in the external mode. This error occurs if there is not a match in the external mode.

The error message is:

```
Start time for scan load cycle in internal mode does not have a match in  
external mode in procedure P. (W27)
```

P is the procedure name.

To correct this violation, adjust the timeplate(s) or procedure so that the internal scan load cycle has a corresponding external cycle starting at the same time.

W28

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In a named capture procedure, the measure_po time in the external mode must match the internal mode. This error occurs if the measure_po time in the external mode does not match the internal mode.

The error message is:

```
Measure_po time in external mode does not match internal mode in procedure  
P. (W28)
```

P is the procedure name.

To troubleshoot this violation, check that there is a measure_po statement in both the internal mode and the external mode. If a measure_po exists in both modes, adjust the timeplate(s) to ensure the measure_po occurs at the same time in both modes.

W29

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In a named capture procedure, the force_pi time in the external mode must match the internal mode. This error occurs if the force_pi time in the external mode does not match the internal mode.

The error message is:

```
Force_pi time in external mode does not match internal mode in procedure  
P. (W29)
```

P is the procedure name.

To troubleshoot this violation, check that there is a force_pi statement in both the internal mode and the external mode. If a force_pi exists in both modes, adjust the timeplate(s) to ensure the force occurs at the same time in both modes.

W30

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, an event on an internal pin is not allowed in the external mode. This warning alerts you that there is an internal pin event in the external mode of the procedure.

The warning message is:

```
Event E N on internal pin is not allowed in external mode of procedure P.  
(W30)
```

E is the type of event, N is the name of the pin to which the event applies, and P is the procedure name.

The tool will ignore the internal pin event in the external mode; however, you can avoid the violation by removing the internal pin event from the external mode.

W31

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A named capture procedure that has an external mode must also have an internal mode. This error occurs when an external mode exists without an accompanying internal mode.

The error message is:

```
Procedure P has an external mode but is missing an internal mode. (W31)
```

P is the procedure name.

To correct this violation, do one of the following:

- Add an internal mode definition to the named capture procedure, or
- Remove the “mode external =” statement and its corresponding “end;” statement from the named capture procedure.

W32

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

A force_pi in a named capture procedure must not occur when a clock is in an on state.

W32 ensures the procedure file is valid.

This DRC checks the force_pi statement in named capture procedures in the procedure file when loading a new procedure file, which can occur during DRC, when the read_procfile command is used, or when the write_patterns command is used with a new procedure file specified on the write_patterns command line.

Messaging

If the procedure file fails W32, the following message displays:

```
A force_pi statement occurs while a clock or control is active in
procedure P (W32).
```

Where P is the name of the procedure file.

Troubleshooting

To correct this violation, edit the procedure file so the force_pi statement does not occur during the time that a clock is in its on state.

W33

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

The named capture procedure in the procedure file cannot be loaded using the -Append_or_timing_update switch in the read_procfile command.

W33 ensures the procedure file is valid.

This DRC checks a new procedure file when it is loaded with the `read_procfile` or `write_patterns` command. A W33 violation occurs when the new procedure file contains a new scan procedure (`load_unload`, `alternate shift`, `master_observe`, and so on) or a new named capture procedure. W33 is only issued for new named capture procedures if the `-Append_or_timing_update` switch is specified with the `read_procfile` command.

Messaging

If the `test_setup` procedure file fails W33, the following message displays:

```
Cannot add new procedure P, procedure ignored (W33)
```

Where P is the procedure name.

Troubleshooting

To correct this violation, do one of the following:

- If the violation is for a named capture procedure, either change the `read_procfile` command option from `-Append_or_timing_update` to `-Replace` or edit the procedure file to remove the named capture procedure.
- If the violation is for a scan procedure, edit the procedure file to remove the scan procedure.
- For both cases, you can return to setup mode and load the procedure file with the `add_scan_groups` command.

W34

Category: Timing

Contexts Supported: `dft -scan`, `patterns -scan`, `patterns -scan_diagnosis`

Default Handling: Error

`report_drc_rules`: Supported

This error occurs when saving patterns if a force event in the external mode conflicts with a pin constraint in a named capture procedure.

The error message is:

```
//Error: Event E N at time T in external mode conflicts with pin  
constraint on same pin in procedure P. (W34)
```

Where E is the event type, N is the name of the event pin, T is the time at which the conflict occurs, and P is the name of the procedure that contains the conflicting pin constraint.

To resolve this issue, either the pin constraint or the named capture procedure must be changed. If this error is ignored, you will not be able to load external mode STIL or WGL patterns back into the tool because the pattern data (using the force event) will not match the pin constraint.

W35

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This error occurs when a pulse-always clock needs to be added to a cycle, and that cycle uses a timeplate that has no pulse timing for the clock.

The error message is:

```
//Error: The following occurred at line N in file F
Pulse-always clock C is not pulsed in T. (W35)
```

Where N is the line number, F is the filename, C is the clock name, and T is the timeplate name.

The W35 rule is an error, and the handling cannot be changed.

W36

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This error occurs when “offstate” statements in the timeplates used by a named capture procedure are defined for a pin that has already been defined as a clock in the dofile. The unnecessary “offstate” statements cause extra pulses in a cycle.

The error message is:

```
// Error: Offstate cannot be used with pin N already defined as a clock
(W36)
```

Where N is the name of the clock pin.

To resolve this error, remove the unnecessary “offstate” statements from the timeplates.

W37

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Each load cycle of a named capture procedure must include a **force_pi** event. The ATPG tools require a **force_pi** event after each scan load. If there is no **force_pi** event in the load cycle of a named capture procedure, the generated test patterns may be incorrect and produce errors when read back into the ATPG tools.

The W37 violation does not override or mask a [W20](#) violation. Therefore, if this error occurs in the first cycle and there is no force_pi event before the first clock pulse, then both a W37 and W20 violation are issued.

The W37 rule can be reduced to a warning or note only if every signal having a force event in a load_unload procedure has a pin constraint defined with a matching value.

The occurrence message is:

```
//Error: A force_pi statement does not exist in a scan load cycle in
Procedure P. (W37-1)
```

Where P is the name of the procedure.

Troubleshooting

Edit the load cycle(s) of the named capture procedure to include a force_pi event.

W38

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule checks whether pulse-always clocks that are pulsed in the external mode of a named capture procedure and are internally connected also pulse in the internal mode of the named capture procedure at the same time.

The error message is:

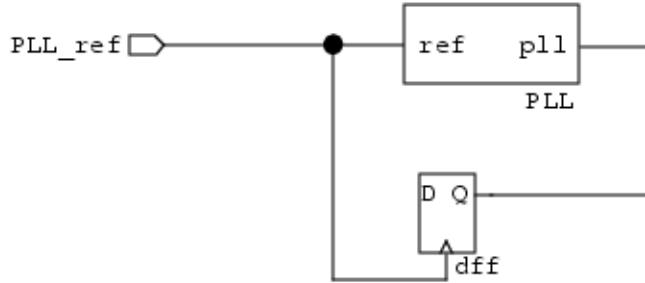
```
// Error: External event E on pulse-always clock at time T needs to have
// matching internal event in procedure P or clock be disconnected from
// internal sequential elements or observe points.
```

Where E is a combination of the event type command and the signal in question. For example, if MyClock was the pulse-always clock, E would be “pulse MyClock”. T is the time at which the conflict occurs, and P is the name of the procedure that contains the unmatched external event.

A PLL clock connected to a sequential cell can also produce a W38 violation. Figure 7-61 shows the type of circuit that would cause a W38 error if there is no matching internal event. For this example, assume that the clock is defined with the following command:

```
> add_clocks 0 PLL_ref -pulse_always
```

Figure 7-61. W38 Reference Circuit



How to Correct W38 Violations

You can correct a W30 violation by either adding a matching internal event for every external mode event, or adding a cut point to disconnect the clock from sequential elements and observe points.

This rule defaults to error handling. If you set the rule to warning handling, the tool attempts to auto-correct the W38 problem by copying external mode event(s) to the appropriate time in the internal mode. If the auto-correct fails (for example, a W35 violation in the internal mode), then the W38 rule's handling is changed to an error by the tool.

W39

Contexts Supported: All contexts

Default Handling: Error

report_drc_rules: Supported

This rule ensures that all test clocks (TCKs) defined in ICL have consistent timing.

When you have an elaborated ICL, regardless of context, W39 is checked during system mode transition, after the read_procfile command. This ensures that the waveforms of all TCK ports, in each timeplate, adhere to these conditions:

- All TCK ports must have the same off state.
- The TCK ports cannot have more than one pulse.
- The waveforms of all TCK ports must either have one pulse or no pulse.

If there is no specific timing for the clock, the tool uses the pulse_clock timing. By default, the W39 DRC triggers an error, but may be set to trigger a warning.

W39 can report three error conditions. The error messages are:

```
// Error: The waveform of TCK port C has multiple pulses. TCK waveform  
cannot have more than one pulse. (W39-1)
```

Where C is the TCK port name.

```
// Error: Inconsistent TCK timings between the 2 TCK ports: C1 and C2.  
All TCK portss must have the same off state. (W39-1)
```

Where C1 and C2 are the TCK port names.

```
// Error: Inconsistent TCK timings between the 2 TCK ports: C1 and C2.  
The waveforms of all TCK ports must either have one pulse or no pulse at  
all. (W39-1)
```

Where C1 and C2 are the TCK port names.

W41

Contexts Supported: All contexts

Default Handling: Error

report_drc_rules: supported

Flags the improper use of timeplates that pulse frequency divided clocks within capture procedures.

The W41 rule check defaults to an error, but its handling can be changed to warning, note, or ignore. The message has the following format:

```
// The following occurred at line 53 in file  
// ..../data/design_with_chains_modified_error.testproc:  
// Error: Event 'pulse user_added_clk_1' at time 100 conflicts with  
// frequency divided clock in procedure my_ncp_proc. (W41-1)
```

Other DRC Messages

DRC generates various messages to indicate the status of various processes and state elements.

Some of them are discussed here.

Transparent Capture Handling Analysis	3052
Oscillation Limitation	3053
RAM Summary Results and Test Capability	3053

Transparent Capture Handling Analysis

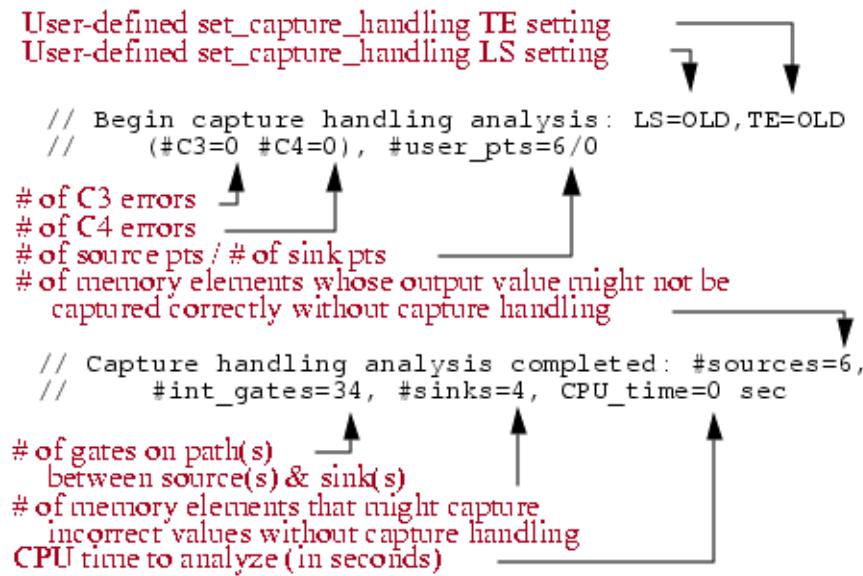
If you issue the set_capture_handling command before running DRC, an analysis message displays upon completion of DRC.

Example

```
// Begin capture handling analysis: LS=OLD, TE=OLD
//      (#C3=0 #C4=0), #user_pts=6/0
// Capture handling analysis completed: #sources=6,
//      #int_gates=34, #sinks=4, CPU_time=0 sec
```

Figure 7-62 describes each component of the messages.

Figure 7-62. Transparent Capture Handling Analysis Messages



Oscillation Limitation

The DRC simulator limits the maximum number of iterations performed to 500 while stabilizing state changes. If state elements continue to change after the maximum number of iterations is reached, the tool displays a message warning that the limit was reached. Additionally, the message reports each state element set to X to limit the iterations. State elements include latches, flip-flops, and feedback buffers.

The warning message is:

```
S forced to X in procedure P at time T due to iteration limit.
```

Where S is the state element, P is the procedure type, and T is the time.

Example

```
// latch1/Q (151) forced to X in procedure load_unload at
//      time 25 due to iteration limit.
```

RAM Summary Results and Test Capability

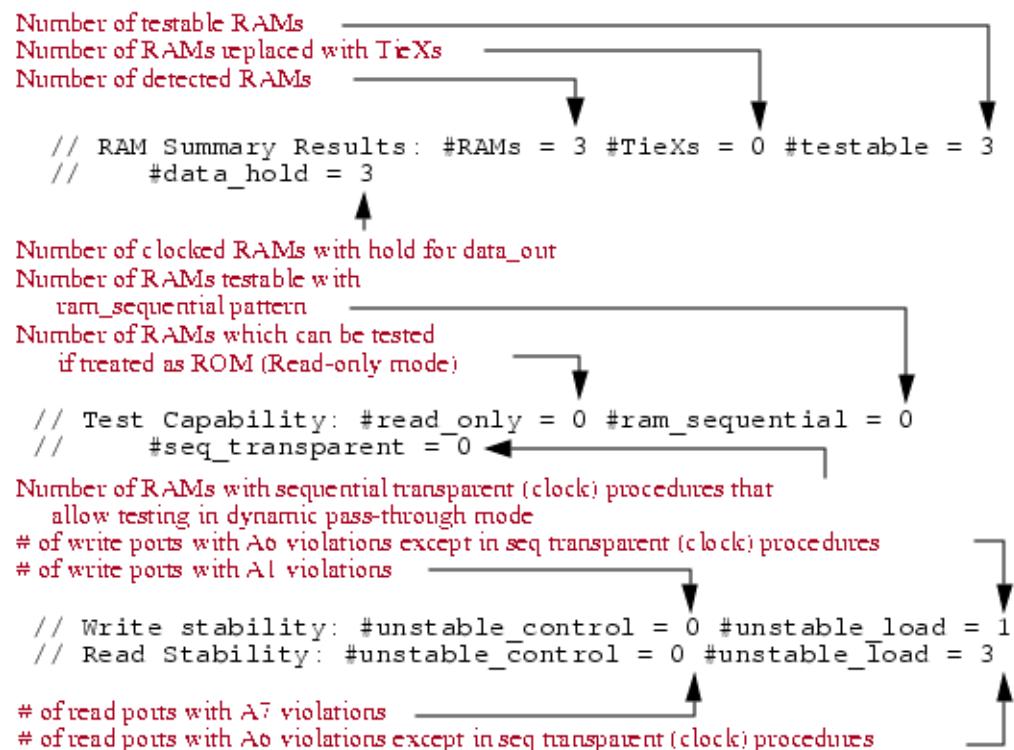
Upon completion of RAM rules checking, DRC displays summary result information as well as test capability information.

Example

```
// RAM Summary Results: #RAMs = 3 #TieXs = 0 #testable = 3
//      #data_hold = 3
// Test Capability: #read_only = 0 #ram_sequential = 0
//      #seq_transparent = 0
// Write stability: #unstable_control = 0 #unstable_load = 1
// Read stability:  #unstable_control = 0 #unstable_load = 3
```

Figure 7-63 describes the information displayed after running RAM rules checking.

Figure 7-63. RAM Summary Results and Test Capability Messages



Chapter 8

Data Models

Tessent Shell has an internal database that is organized into data models which are broad categories of design information (such as hierarchical design data). Each of the data models contains one or more types of design objects (such as pins and modules), and each design object has a list of attributes (such as an ID or bit-width of a bus).

For additional information about the design data models and how to work with them, refer to “[Design Data Model, Attributes, and Introspection](#)” in the *Tessent Shell User’s Manual*.

You can access the pre-defined attributes on those design objects using the *_attribute* tool commands listed in [Table 9-1](#) on page 3159. Those commands also provide a way for you to register new user-defined attributes for those design objects in the database.

The following sections describe the various data models, the types of objects they can contain, and the attributes associated to the objects:

Hierarchical Design Data Model	3056
Module	3058
Instance	3069
Port	3080
Pin	3094
Net	3099
Pseudo_port	3104
Flat Design Data Model.....	3107
Flat Design Object Types	3108
ICL Data Model.....	3118
ICL Object Types	3119
Scan Data Model	3144
Scan Element Object Type	3144
Scan Chain Family Object Type	3150
Scan Mode Object Type	3151
Test Point Data Model.....	3151
Configuration Data Model	3153
config_csv_wrapper	3153
config_data_wrapper	3154
config_property	3155
config_repeatable_property	3155
config_wrapper	3156

Hierarchical Design Data Model

The hierarchical design data model contains six different object types: module, instance, port, pin, net, and pseudo_port.

As suggested by [Figure 8-1](#), instances, pins, and nets are hierarchical objects that exist relative to the current design. Ports and modules, shown in [Figure 8-2](#), are non-hierarchical objects.

The figures also show the attributes that can be associated with the design objects.

The *current design* is the design specified by the set_current_design command.

Figure 8-1. Tessent Shell Hierarchical Design Objects

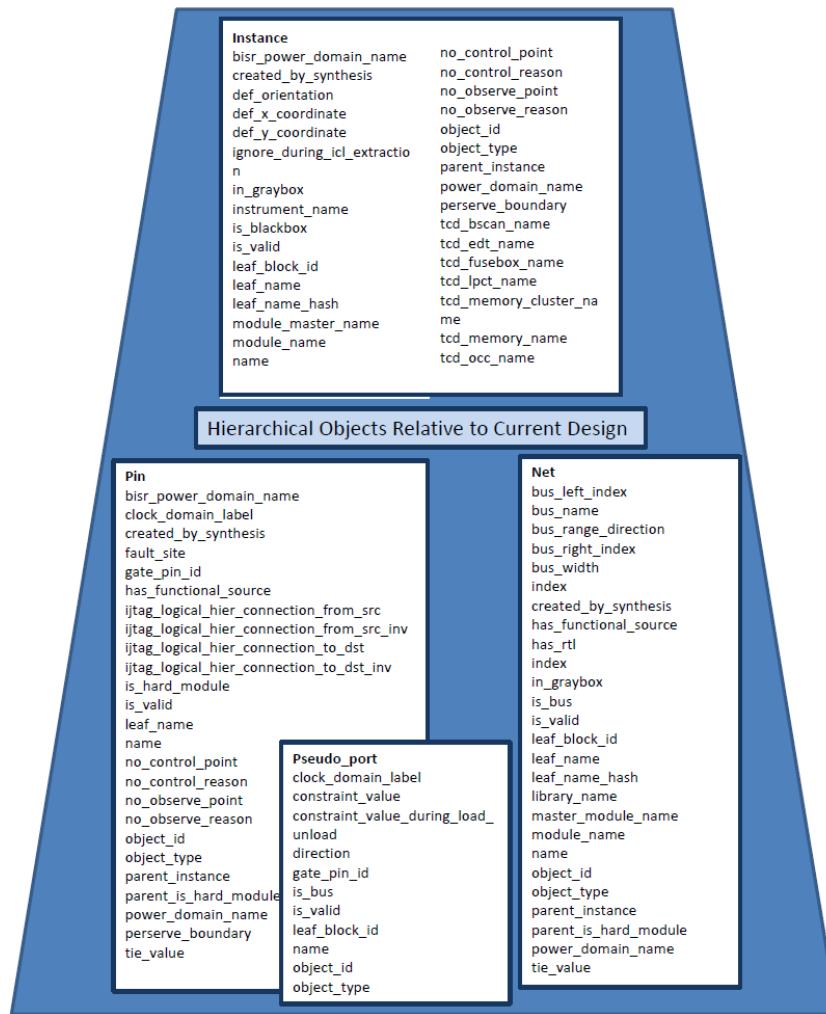


Figure 8-2. Tessent Shell Non-Hierarchical Design Objects

The six hierarchical design object types are described on the following reference pages:

[Module](#), [Instance](#), [Port](#), [Pin](#), [Net](#), and [Pseudo_port](#).

Reference information about attributes is provided in section “[Attributes](#)” on page 3159.

Module	3058
Instance	3069
Port	3080
Pin	3094

Net	3099
Pseudo_port	3104

Module

A module definition can come from a Verilog module, a VHDL entity, a Tessent library model, or it can be a built-in primitive. A module has a simple non-hierarchical name.

The module object type interacts with attributes as follows:

- Attributes are automatically visible as read-only attributes on instances if the same attribute is not also registered on the instance object type. Such attributes remain stored on the module definition and not on the instance itself
- Attributes with the `applies_to_child_instances` option set to On are visible to all child instances below the module.

Built-In Attributes

The module object type has the following attributes:

active_child_scan_mode	<p>When a child core exposes scan chain segments from several scan modes, this attribute makes the existing segments from a specific child scan mode active. During DRC, these active segments are the ones that get traced if tracing is enabled.</p> <p>Scan segments of the “active” child scan mode get the “state” attribute “usable”, while segments from other child scan modes are attributed with the “state” “unusable_child_mode”.</p> <p>Accordingly, during analysis, when creating scan chain families and/or adding scan modes, this attribute is also used to control their populations. If the tool cannot uniquely resolve the active scan mode for an instance by using the rules below, then the “state” attribute is set to “unresolved_child_mode”.</p> <p>The tool determines the effective “active_child_scan_mode” attribute for scan segments of a multi-mode instance using the following rule sequence:</p> <ol style="list-style-type: none">1. The mode corresponding to the “active_child_scan_mode” if it is set for the instance and matches one of its child modes.2. The mode of type “external” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “external” exists.3. The mode of type “unwrapped” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “unwrapped” exists.
------------------------	---

cell_type	String that reports characteristics of the cell. This attribute is initially null and is populated by the set_current_design command with information extracted from the cell library files. For example, any cell described as a pad in the library will have cell_type set to “pad”. If your cell library includes the scan mapping information, the non-scan flip-flop will be identified with a nonscan_dff type and the scan equivalent flip-flops will have mux_scan_dff type. See the read_cell_library command description for information on how the cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command if your cell library file only includes the ATPG cell modeling information. <i>Tessent Cell Library Manual</i> .
def_x_coordinate_list	For the top module of the current design, this attribute is a list of all the x coordinates from a specified def file when you execute the read_def command.
def_y_coordinate_list	For the top module of the current design, this attribute is a list of all the corresponding y coordinates of the die from a specified def file when you execute the read_def command.
def_xl_coordinate	For the top module of the current design, the lower left x coordinate from a specified def file when you execute the read_def command. For other modules, the lower left x coordinate from a specified LEF file when you execute the read_lef command.
def_yl_coordinate	For the top module of the current design, the lower left y coordinate from a specified def file when you execute the read_def command. For other modules, the lower left y coordinate from a specified LEF file when you execute the read_lef command.
def_xh_coordinate	For the top module of the current design, the upper right x coordinate from a specified def file when you execute the read_def command. For other modules, the upper right x coordinate from a specified LEF file when you execute the read_lef command.
def_yh_coordinate	For the top module of the current design, the upper right y coordinate from a specified def file when you execute the read_def command. For other modules, the upper right y coordinate from a specified LEF file when you execute the read_lef command.
design_id	A string attribute that is set to reflect its design_id. It is set by read_design when reading a design. It also gets set after elaboration on the current design to the value returned by get_context -design_id when the value is not null.

design_level	A string attribute that is set to reflect its design_level. It is set by read_design when reading a design. It also gets set on the current design when the set_design_level command is run. The value is set to one of the following: <ul style="list-style-type: none">• chip• physical_block• sub_block
exclude_from_concatenated_netlist	Boolean that specifies that the given module is a hard macro and should not be included in the concatenated netlist.
exclude_from_synthesis	Boolean that specifies that the given module is excluded from any synthesis. During ICL extraction and other tasks, the combinational path between the IJTAG port of the modules having a corresponding ICL module is traced and any module encountered along the way is synthesized. The pre-tracing step that does the synthesis performs a structural tracing of the input or output cone of a port. In the opposite case, following the flat model tracing only follows the controllable paths. Therefore in some cases the synthesis on the modules is not required and may take a long time, for example, for large memories that were not defined as memories. The exclude_from_synthesis attribute can be used to shorten the pre tracing time in that case.
file_path_name	String that specifies the pathname to the file where a given module is defined. This attribute is null for created modules or modules without a definition.
format	Contains the value of the -format option as it was specified by the read_verilog or read_vhdl command when the module was read in. This value is only defined for modules with the “language” attribute set to “verilog” or “vhdl”; if the language attribute is set to “tessent_lib”, an empty string is returned. See the “language” attribute for more information. For no_rtl and created modules, “2001” is the value for “verilog”.
has_async_set_reset	Boolean that is true when a module has an asynchronous set port (port with active_high_set or active_low_set pin function) and/or an asynchronous reset port (port with active_high_reset or active_low_reset pin function).
has_def	Boolean that indicates whether or not the DEF file of the module was read in.
has_no_definition	Boolean that is true when a module does not have a definition, but a blackbox model was created during elaboration from its instantiation. You cannot have this type of module before creating the flat module unless you identify the module as a blackbox using the add_black_boxes command.

ijtag_is_icl_module	Boolean that identifies a design module as an ICL module. This attribute is only set in the patterns -ijtag context.
instrument_name	String that is set to the ICL module name it matches when its ijtag_is_icl_module attribute is true. The default is null.
is_blackbox	Boolean that returns true if the module was blackboxed in one of the following manners: <ul style="list-style-type: none">• The add_black_boxes -module command was issued on the module• The add_black_boxes -auto switch is used• The module was read using read_verilog -interface_only or read_vhdl -interface_only A false value is returned otherwise.
is_created	Boolean that specifies whether the module was created using the create_module command in insertion mode. In -rtl mode, this attribute is set as soon as the module is created and remains set until the insertion state is left. In -no_rtl mode, this attribute is set once when the module is created and never reset.
is_elaborated	Boolean that specifies whether the module is part of an elaborated design. When set to true, the module is part of the elaborated design, that is, it is either the current design module or there is at least one instance of this module.
is_graybox	Boolean that is set to true for the top module loaded with read_design -view graybox. When reading a graybox module of a child physical block, the exclude_from_concatenated_netlist attribute is set on the module such that if you call the write_design command, the child physical graybox netlist is not concatenated within the parent physical netlist. When using the write_design -graybox command, elements that the tool finds within the child physical graybox may need to be preserved in the parent physical region's graybox. This typically happens when a clock is sourced from a PLL found inside the child physical region. The is_graybox attribute set on that module is used by the write_design -graybox command to ignore the exclude_from_concatenated_netlist attribute for that module.

is_hard_module	<p>Boolean that specifies the module cannot be edited when set to true. This attribute is automatically set to true for all modules with type = “cell” or “primitive”. This attribute is set for design modules in the dofile using the following command:</p> <pre>set_attribute_value -name is_hard_module.</pre> <p>This property has applies_to_child_instances set to On by default. The setting of this attribute also has an effect on the is_non_editable instance attribute.</p>
is_incomplete	<p>Boolean that specifies that the module cannot be edited and/or written out when set to true. This attribute is set to true when the module was read with the read_verilog or read_vhdl -interface_only commands, or when the module includes RTL constructs that are not supported by the set_context -no_rtl option. The is_hard_module attribute automatically returns true for any module with the is_incomplete attribute set to true. To write out a module with only its port and parameter definition, read it in normally and write it back out using the write_design -interface option.</p>
is_internal_scan_only	Boolean that specifies that the module is internal scan only.
is_internal_scan_only_reason	<p>String that specifies the reason the instance is internal scan only as follows:</p> <ul style="list-style-type: none"> • imported_from_tcd_scan — The object was inferred to be internal scan only because it has a Core/ScanTCD description and the property internal_scan_only is equal to 1. • user_specified — The instance is internal scan only because the specified_internal_scan attribute was set to true on this instance or the instance inherited the true value from an instance in its ancestry. <p>The values are reported only when is_internal_scan_only is true.</p>
is_modified	<p>Boolean that specifies that a module read in from a file was edited during the insertion mode.</p> <p>In -rtl mode, this attribute is set as soon as the module is modified and remains set until the insertion state is left.</p> <p>In -no_rtl mode, the attribute is set once when the module is edited and never reset.</p> <p>See the get_modules command description for usage examples.</p>
is_non_scannable	Boolean that specifies that the module is non-scannable.

is_physical_module	Boolean that specifies that the module is a physical block. It gets set during design elaboration if the module is found in an open TSDB and its level was defined as chip or physical_block when its TSDB was created.
is_power_always_on_cell	Boolean that is set to true when the module is specified as an always_on cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_isolation_cell	Boolean that is set to true when the module is specified as an isolation cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_level_shifter_cell	Boolean that is set to true when the module is specified as a level shifter cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_retention_cell	Boolean that is set to true when the module is specified as a retention cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_synthesized	Boolean that specifies that the given module was synthesized when going to analysis mode by the quick synthesis engine.
is_unsaved	Boolean that specifies that a module that was either modified or created within the insertion mode of the dft context was not yet written out. This attribute is set as soon as the module is modified or created and is reset as soon as the module is saved using the write_design command. See the get_modules command description for examples of its usage.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
language	String that specifies the supported library language. Set to “tessent_lib” for Tessent library models. Set to “verilog” for Verilog modules and Verilog primitives. Set to “vhdl” for VHDL entities.
library_name	String that specifies the logic name of the library containing the module definition. All commands that accept a module name look for the module in the default library as specified by the set_logical_design_libraries command. You can use the get_modules command to access modules from different libraries. Libraries are only supported in -rtl sub-contexts. In -no_rtl contexts, one library exists and it is named “work”.
line_number	Integer that specifies a line number in the file where a given module is defined. This number is 0 for created modules and for modules without a definition.

master_name	String that specifies the real name of the module as it was defined independently of the parameterized view.
master_object_id	An attribute that returns the object_id of the master module associated with this parameterized module. If this module is not a parameterized view, it returns the object_id of this module.
name	<p>String that specifies the name of the module. Modules are defined from the following sources:</p> <ul style="list-style-type: none">• read_verilog files• read_vhdl files• create_design and copy_design commands• read_cell_library files• primitives have built-in names <p>For parameterized modules, a unique name is created for each parameterized view by appending to the master_name the string @PV# where # is an integer starting from 1 and incremented for each parameterized view. Use the get_module_parameter_list and get_module_parameter_value commands to introspect the parameter values.</p>
no_control_point	An attribute that specifies that no control point can be inserted on any pin, port or instance in the specified module when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.

no_control_reason	<p>String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>The possible values for the tool identified locations are:</p> <ul style="list-style-type: none">• cell_internal - all the gate pins inside the library cells.• clock_cone - including set and reset line.• power-aware - input pins of power cells (isolation cell, level shifter)• scan_path - including sub_chain, shift_register and any logic between scan in and scan out.• tristate_enable - including tristate logic and bidirectional gates.• constant_value_during_capture - including all the gates that have constant value during capture. This can be determined either by the add_input_constraints command or test procedure. <p>The possible values for the user-specified locations are:</p> <ul style="list-style-type: none">• explicitly_specified - can be specified using the “add_nptest_point” command or “set_attribute_value” command for and no_observe_point attributes.• cross_clock_domain - can be specified using “set_test_point_analysis_option -exclude_cross_domain_paths” command.• xbounding - can be specified using “set_test_point_analysis_options -allow_in_xbounding_regions” command or “analyze_xbounding” command in dft –scan context.• multicycle_path - can be specified by reading SDC file .• false_path - can be specified by reading SDC file.• excluded_clock_domain - can be specified using “set_test_point_analysis_option -exclude_clock_domains” command.
no_observe_point	An attribute that specifies that no observe point can be inserted on any pin, port or instance in the specified module when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.

no.observe_reason	<p>String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>The possible values for the tool identified locations are:</p> <ul style="list-style-type: none"> • cell_internal — All the gate pins inside the library cells. • clock_cone — Including set and reset line. • power-aware — Output pins of power cells (isolation cell, level shifter) • scan_path — Including sub_chain, shift_register and any logic between scan in and scan out. • tristate_enable — Including tristate logic and bidirectional gates. • constant_value_during_capture — Including all the gates that have constant value during capture. This can be determined either by the add_input_constraints command or test procedure. <p>The possible values for the user-specified locations are:</p> <ul style="list-style-type: none"> • explicitly_specified — Can be specified using the “add_notest_point” command or “set_attribute_value” command for and no.observe_point attributes. • cross_clock_domain — Can be specified using “set_test_point_analysis_option -exclude_cross_domain_paths” command. • xbounding — Can be specified using “set_test_point_analysis_options -allow_in_xbounding_regions” command or “analyze_xbounding” command in dft –scan context. • multicycle_path — Can be specified by reading SDC file. • false_path — Can be specified by reading SDC file. • excluded_clock_domain — Can be specified using “set_test_point_analysis_option -exclude_clock_domains” command.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "module: <i>integer</i> "
object_type	Returns the object data type “module”.

power_domain_island	Returns the name of the power domain island associated to the current design. This attribute is null for all other modules. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain Island , refer to the <i>Tessent Glossary</i> .
power_domain_name	Returns the name of the power domain associated to the current design. This attribute is null for all other modules. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain , refer to the <i>Tessent Glossary</i> .
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins on all instances of the module will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
simulation_function	String that specifies the function as learned by simulation. Refer to “ Model-Level Attribute Descriptions ” in the <i>Tessent Cell Library Manual</i> for complete information.
specified_internal_scan_only	An attribute used to specify that the instance of the module are to be forced to remain internal during wrapper analysis. Dedicated isolation wrapper cells will be inferred to isolate the input and output pins of the instances if any interacts with the primary input and output ports.
sync_cell_length	For posedge and negedge synchronizer cells, this attribute is an integer that specifies the number of DFFs in the cell. For all other cell types, this attribute is an empty string.

synthesize_before_analysis	<p>Boolean that specifies that the given module is to be synthesized when going to analysis mode.</p> <p>During ICL extraction, the combinational path between the IJTAG port of the modules having a corresponding ICL module is traced and any module encountered along the way is synthesized. If you use a test_setup procedure during ICL extraction, you must set this attribute on all RTL modules which must be synthesized in order for the test setup simulation to propagate its value to the entire ICL network as the pretracer does not traverse sequential cells. The quick synthesis engine uses a cell library defined in the build tree at the given location “\$TESSENT_HOME/lnx-x86/lib64/hdleng/lib”. Both Tessent and Verilog cell library models are supplied, namely “dft_sim.atpg” and “dft_sim.v”. The tessent libraries are automatically loaded when synthesis is used.</p>
tcd_bscan_name	<p>A string attribute that gets set on every module for which there is a matched TCD BoundaryScan description (see “Tessent Core Description” on page 3755).</p> <p>See “set_design_sources -format tcd_bscan” and set_module_matching_options commands for more information about this matching process.</p>
tcd_fusebox_name	<p>A string attribute that gets set on every module for which there is a matched TCD FuseBoxInterface description (see “Tessent Core Description” on page 3755).</p> <p>See “set_design_sources -format tcd_memory” and set_module_matching_options commands for more information about this matching process.</p>
tcd_memory_cluster_name	<p>A string attribute that gets set on every module for which there is a matched TCD MemoryCluster description (see “Tessent Core Description” on page 3755).</p> <p>See “set_design_sources -format tcd_memory_cluster” and set_module_matching_options commands for more information about this matching process.</p>
tcd_memory_name	<p>A string attribute that gets set on every module for which there is a matched TCD Memory description (see “Tessent Core Description” on page 3755). This attribute is also set on memory modules that are <i>inside</i> another memory.</p> <p>See “set_design_sources -format tcd_memory” and set_module_matching_options commands for more information about this matching process.</p>

type	<p>Set to “cell” for the following modules:</p> <ul style="list-style-type: none"> • Models read in with <code>read_cell_library</code>. • Verilog modules surrounded by the <code>`celldefine</code>/ <code>`endcelldefine</code> directives and read in with <code>read_verilog</code>. <p>Set to “primitive” for the following modules:</p> <ul style="list-style-type: none"> • Verilog primitives used in the design. <p>Set to “design” for all other modules.</p>
vhdl_architecture	<p>String that reports the architecture of vhdl modules. It is null for modules of other languages. All commands that accept a module name return the vhdl entity associated with the default architecture which is the last compiled version. Use get_modules to access modules with the non default architecture.</p>

Inheritance

Built-in inherited attributes:

- None

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Module Specification

A module object (`module_object`) is specified by a module name which is a string. For example: “modA”.

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_common_parent_instance](#)
[get_modules](#)
[get_name_list](#)
[get_ports](#)
[report_attributes](#)

Instance

An instance is a single instantiation of a module. No instance object exists until the `set_current_design` command is executed.

The instance object type interacts with attributes as follows:

- **Module** attributes are automatically visible as read-only attributes on corresponding instances if the instance object type does not have the same attribute registered.
- Attributes defined with the `applies_to_child_instances` option set to On automatically become visible to all child instances of the instance object.

Built-In Attributes

active_child_scan_mode	<p>When a child core exposes scan chain segments from several scan modes, this attribute makes the existing segments from a specific child scan mode active. During DRC, these active segments are the ones that get traced if tracing is enabled.</p> <p>Scan segments of the “active” child scan mode get the “state” attribute “usable”, while segments from other child scan modes are attributed with the “state” “unusable_child_mode”. Accordingly, during analysis, when creating scan chain families and/or adding scan modes, this attribute is also used to control their populations. If the tool cannot uniquely resolve the active scan mode for an instance by using the rules below, then the “state” attribute is set to “unresolved_child_mode”.</p> <p>The tool determines the effective “active_child_scan_mode” attribute for scan segments of a multi-mode instance using the following rule sequence:</p> <ol style="list-style-type: none">1. The mode corresponding to the “active_child_scan_mode” if it is set for the instance and matches one of its child modes.2. The mode of type “external” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “external” exists.3. The mode of type “unwrapped” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “unwrapped” exists.
bisr_power_domain_name	An attribute that can be set on memory instance to associate a BISR power domain name to a memory. You typically do not need to use this attribute when you read in a UPF or CPF file as the <code>power_domain_name</code> attribute will be used to infer the power domain of the memories. You use the <code>bisr_power_domain_name</code> attribute if you want to have more BISR power than those described in the UPF/CPF file. You use this attribute on Pins to map specific BISR chains on subblock to power domain at the current level as documented in the <code>bisr_segment_order_file</code> section.

created_by_synthesis	Boolean that indicates that the instance was created by quick synthesis. When an instance has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
def_orientation	Contains the DEF orientation of the flip-flop, memory, or sub-physical region extracted by the read_def command. The attribute can have the following values: N, S, E, W, FN, FS, FW, FN.
def_x_coordinate	The x coordinate of an instance extracted from a specified def file when you execute the read_def command.
def_y_coordinate	The y coordinate of an instance extracted from a specified def file when you execute the read_def command.
ignore_during_icl_extraction	When set to true, even if the instance has an associated ICL module during ICL extraction, the instance is ignored and not reported in the extracted ICL. This attribute is useful to allow extracting the ICL for the portion of the network that is correct while other instances still have issues to resolve. The generated ICL module will have the <code>tessent_ignored_design_instances</code> attribute defined referencing all design instances that were ignored.
in_graybox	Set to true when the instance is included in the graybox netlist. Set to false when the instance is excluded from the graybox netlist, which is the default for all the instances in the design. You can change the attribute value with the analyze_graybox or set_attribute_value command. For more information about this attribute, refer to the analyze_graybox command description.
instrument_name	A string attribute that is set on instances of modules for which there is a matched TCD instruments description. If a TCD instruments instance is instantiated below another TCD instruments instance, the attribute is only set on the higher instance. See set_design_sources -format icl and set_module_matching_options commands for more information about this matching process.
is_blackbox	Boolean that specifies that the instance was black boxed with add_black_boxes -instance or that the module of the instance was black boxed as described for the <code>is_blackbox</code> attribute of the module object type. The default is false.
is_internal_scan_only	Boolean that specifies that the instance is internal scan only.

is_internal_scan_only_reason	<p>String that specifies the reason the instance is internal scan only as follows:</p> <ul style="list-style-type: none"> • imported_from_tcd_scan — The object was inferred to be internal scan only because it has a Core/Scan TCD description and the property internal_scan_only is equal to 1. • user_specified — The instance is internal scan only because the specified_internal_scan attribute is set to true on this instance or the instance inherited the true value from an instance in its ancestry. • user_specified_from_net — The instance is internal scan only because the specified_internal_scan attribute was set on a net which happens to be an RTL register. The instances happen to be the flop-flop instance that the tool created during quick synthesis for that RTL register. <p>The values are reported only when is_internal_scan_only is true.</p>
is_non_editable	Boolean that specifies that the instance is non-editable.
is_non_editable_reason	<p>A string that specifies why the instance is non-editable as follows:</p> <ul style="list-style-type: none"> • below_hard_module — The instance is below an instance which has is_hard_module equal to 1 • hard_module — The instance has is_hard_module equal to 1 • created_by_quick_synthesis — The instance was created by quick synthesis • matched_icl_module — The module of the instance has a matched ICL module • below_matched_icl_module — The instance is below a module that has a matched ICL module
is_non_scannable	Boolean that specifies that the instance is non-scannable.

is_non_scannable_reason	A string that specifies why the instance is non-scannable as follows: <ul style="list-style-type: none">• inferred_from_drc — The object was inferred to be non-scannable during DRC either because it had an S-rule violation or it is part of a hard_module yet excluded from the traced scan chains.• imported_from_tcd_scan — The object was inferred to be non-scannable because it has a Core/Scan TCD description and the property drc_from_boundary_only is equal to 1.• imported_from_icl — The instance was inferred to be non-scannable because it has a matched ICL description and the attribute keep_active_during_scan_test is not specified as false.• imported_from_bscan — The instance was inferred to be non-scannable because it is part of the boundary scan chain.• missing_scan_model — The instance was inferred to be non-scannable because it does not have a scan flop equivalent mapping. This is only happening in dft –scan context.• user_specified — The instance was specified to be non-scannable using the add_nonscan_instances command.• user_specified_from_net — The instance is non scannable because the add_nonscan_instances -rtl_reg command was used on a net which happens to be an RTL register. The instances happens to be the flop-flop instance that the tool created during quick synthesis for that RTL register. If is_non_scannable is false, then is_nonscannable_reason returns an empty string.
is_power_always_on_cell	Boolean that is set to true when the instance is specified as an always_on cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_isolation_cell	Boolean that is set to true when the instance is specified as an isolation cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_level_shifter_cell	Boolean that is set to true when the instance is specified as a level shifter cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_retention_cell	Boolean that is set to true when the instance is specified as a retention cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.

leaf_block_id	The leaf_name of instances located inside generate blocks have the generate block ids joined by periods prepended to their instantiation name. This attribute returns the block_id part of the leaf_name. It is null for instances that are not inside generate blocks.
leaf_name	String that specifies the name of the instance (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_leaf_name attribute.
leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with <i>parent_module.leaf_name_hash.pin_leaf_name</i> entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage. Note that the value of this attribute always refer to the pre_synthesis_name just like the pre_synthesis_leaf_name attribute.
master_module_name	String that specifies the value of the master_name on the module object associated with the instance.
module_name	String that specifies the value of the name of the module object associated with the instance.
name	String that specifies the hierarchical name of the instance relative to the current design. The name of every instance is unique. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_name attribute.
no_control_point	An attribute that specifies that no control point can be inserted on any pin or instance in the specified instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent instances.

no_control_reason	<p>String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.</p>
no_observe_point	<p>An attribute that specifies that no observe point can be inserted on any pin or instance in the specified instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent instances.</p>
no_observe_reason	<p>String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.</p>
object_id	<p>The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "instance: <i>integer</i>"</p>
object_type	Returns the object data type "instance".
parent_instance	String that specifies the instance name in which the instance exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
power_domain_island	Returns the name of the power domain island associated to the instance. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain Island , refer to the Tessent Glossary .

power_domain_name	Returns the name of the power domain associated to the instance. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain , refer to the <i>Tessent Glossary</i> .
pre_synthesis_leaf_name	String that specifies the original RTL leaf name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the leaf_name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_leaf_name will only ever be different from the leaf_name when the instance is inside a generate block that the tool synthesized during quick synthesis.
pre_synthesis_leaf_block_id	String that shows which generated block instantiation is the current focus of the tool. This attribute is used to detect and compute instances that are from generated blocks.
pre_synthesis_leaf_name_hash	String that shows where the parameter of the generated block is located. This attribute is used to detect and compute instances that are from generated blocks.
pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name will only ever be different from the name when the instance has one or more generate block in its ancestry that the tool synthesized during quick synthesis
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins on the instance will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.

specified_internal_scan_only	An attribute used to specify that the instance is to be forced to remain internal during wrapper analysis. Dedicated isolation wrapper cells will be inferred to isolate the input and output pins of the instances if any interacts with the primary input and output ports. When the attribute is not specified on an instance it inherits its value from its associated module when the attribute is specified on the module. If it is not specified on the module, it inherits its value from its parent instance.
specified_wrapper_type	An attribute that forces the wrapper type of a flop or clock gater instance. Available values are: off, input, output, and generic. The default is off. When set or left to default to off, the wrapper_type value of each flop and clock gater instance is computed by the analyze_wrapper_cells command. Special treatment is performed on instances of OCCs. See the inferred_from_clock_source value in the wrapper_type_reason attribute description below. When you set the specified_wrapper_type attribute to input, output, or generic, the specification is kept during the analysis. If you used the input value, the analyze_wrapper_cells command will automatically find all flip-flops fanning out from the instance you forced to input type and infer them as output wrapper type. The result of the analyze_wrapper_cells command, which combines the specified_wrapper_type values with the inferred values, are reflected in the wrapper_type and wrapper_type_reason attributes described below.
tcd_bscan_name	A string attribute that is set on instances of modules for which there is a matched TCD BoundaryScan description (see “ Tesson Core Description ” on page 3755). If a tcd BoundaryScan instance is instantiated below another tcd BoundaryScan instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_bscan and set_module_matching_options commands for more information about this matching process.
tcd_edt_name	A string attribute that is set to the name of a core instance when the EDT instrument core instance is associated with a design instance. This ensures boundary preservation during flattening.
tcd_fusebox_name	A string attribute that is set on instances of modules for which there is a matched TCD FuseBoxInterface description (see “ Tesson Core Description ” on page 3755). If a tcd FuseBoxInterface instance is instantiated below another tcd FuseBoxInterface instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_fusebox and set_module_matching_options commands for more information about this matching process.
tcd_lpct_name	A string attribute that is set to the name of a core instance when the LPCT core instance is associated with a design instance. This ensures boundary preservation during flattening.

tcd_memory_cluster_name	A string attribute that is set on instances of modules for which there is a matched TCD MemoryCluster description (see “ Tessent Core Description ” on page 3755). If a TCD Memory or MemoryCluster instance is instantiated below another TCD MemoryCluster instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_memory_cluster and set_module_matching_options commands for more information about this matching process.
tcd_memory_name	A string attribute that is set on instances of modules for which there is a matched TCD Memory description (see “ Tessent Core Description ” on page 3755). If a TCD Memory instance is instantiated below another TCD Memory instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_memory and set_module_matching_options commands for more information about this matching process.
tcd_occ_name	A string attribute that is set to the name of a core instance when the OCC core instance is associated with a design instance. This ensures boundary preservation during flattening.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.
wrapper_type	An attribute used to reflect the resulting wrapper type of flop and clock gater instances based on the specified_wrapper_type attribute value you set, special processing of the OCC modules, and the result of the analysis after the analyze_wrapper_cells command is issued. See the specified_wrapper_type attribute description above for more information. The instances of type input have their scan enable held on during the input test modes. The cells of type output have their scan enable held on during the output test modes. The cells of type generic do not have their scan enable held on during any mode type. The allowed values of this attribute are input, output, generic, or an empty string.

wrapper_type_reason	An attribute that describes why the wrapper has a given type. <ul style="list-style-type: none">• inferred_from_analysis — Indicates the type was inferred by the analyze_wrapper_cells command.• inferred_from_clock_source — Means the instance is an OCC, the fast_clock port of the OCC is not directly sourced by a PI (that is, a PLL or clock divider), and the OCC drives at least one flop of type input or output wrapper. The wrapper_type is always generic when wrapper_type_reason is inferred_from_clock_source.• user_specified — Indicates the specified_wrapper_type attribute was explicitly set on the flop or clock gater instances.
---------------------	---

Inheritance

Built-in attribute inheritance from associated module objects:

- cell_type
- has_no_definition
- ijtag_is_icl_module
- is_created
- is_hard_module
- is_incomplete
- is_modified
- is_synthesized
- is_unsaved
- language
- library_name
- preserve_boundary
- type
- vhdl_architecture

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Instance Specification

An instance object (instance_object) is specified by a complete hierarchical instance pathname. For example: “/u1/u2/u3”.

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_common_parent_instance](#)
[get_name_list](#)
[report_attributes](#)

Port

A port is an object on a module and can have an input, output or inout direction. Ports have a simple non-hierarchical name.

One port object exists for each bit of a bussed port so every bit of a bussed port can be attributed independently.

The port object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types. These attributes are stored on the associated port of the pins and not on the pin object itself.

Built-In Attributes

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
bisr_function	String that reflects the function of the design port for Tessent MemoryBIST's BISR. Legal values for this attribute are: Reset, MemoryDisable, Select.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.

bus_name	String that specifies the name of the bus without the index. $\langle \text{name} \rangle = \text{bus_name} [\text{index}]$ when <code>is_bus</code> is true; otherwise, <code>bus_name</code> is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the <code>to</code> and <code>downto</code> <code>BIT_VECTOR</code> types in VHDL.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether <code>left_index</code> is smaller, larger than, or equal to <code>right_index</code> .
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the <code>DFT_C1</code> , <code>DFT_C2</code> , and <code>DFT_C3</code> DRC rules. It is attached to every port on which a clock was defined. The string contains the label associated to the clock defined on the port. It is also set by <code>DFT_C6</code> on every node between the clock and the sequential elements.
constraint_value	String that specifies that the port’s primary input was set to a constant with the <code>add_input_constraints</code> command. When set to <code>C0</code> , <code>CT0</code> , <code>Cl</code> , <code>CT1</code> , <code>CX</code> , or <code>Cz</code> , this attribute displays <code>0</code> , <code>0</code> , <code>1</code> , <code>1</code> , <code>x</code> , or <code>z</code> , respectively. The default is null.
constraint_value_during_load_unload	String that specifies a constant value for a top-level port during <code>load_unload</code> test procedure retargeting. Possible values are: <code>0</code> , <code>1</code> , <code>X</code> , or <code>Z</code> . Constraining the value of the top-level port enables tracing of signals from the core level to the chip level.
created_by_synthesis	Boolean that indicates that the port was created by synthesis. When a port has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
def_x_coordinate	The x coordinate of a top-level port extracted from a specified def file when you execute the <code>read_def</code> command.

def_y_coordinate	The y coordinate of a top-level port extracted from a specified def file when you execute the read_def command.
direction	String that specifies the direction of the port. Possible values are as follows: <ul style="list-style-type: none">• buffer (VHDL)• inout (Verilog and VHDL)• input (Verilog and VHDL)• linkage (VHDL)• output (Verilog and VHDL)
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the <code>-display_in_gui</code> option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the dynamic DFT signal.

function	<p>String that either reports the function of the port on cell modules or reports the presence of the power or ground attributes on ports of the current_design. For a list of possible values, see “Function Attribute Values” on page 3090.</p> <p>For cell modules, this attribute is initially null and is set by the set_current_design command from information extracted from the cell library files. For information about how cell_type information can be loaded, refer to the read_cell_library command. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. See the table in the Pin Attributes section of the <i>Tessent Cell Library Manual</i> for a complete list of available function values.</p> <p>For a top-level module, this attribute is initially null and you can set one of the following values on an input/inout port with the set_attribute_value command:</p> <ul style="list-style-type: none">• power — Constrains power input/inout ports during circuit setup and DRC.• ground — Constrains ground input/inout ports during circuit setup and DRC. <p>For more information about using these values, refer to “Power and Ground Ports Exclusion from the Pattern Set” in the <i>Tessent Scan and ATPG User’s Manual</i>.</p>
gate_pin_id	Returns the id of a gate_pin when the pin has an associated gate_pin inside the flat model. Exists for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the port is of direction output and the port net is undriven or tied.

ignore_during_icl_extraction	Ports of the current design with this attribute set to true are ignored during ICL extraction. This is useful, for example, if an ICL instance has connections to primary ports as well as to other ICL instances, but the ICL shall contain only the connections between the instances. This attribute does not have any effect on ports of other modules besides the current design.
ignore_for_graybox	Set to true when an output port is to be excluded from graybox analysis. By default, this attribute is set to false for all output ports (considered by the graybox analysis backtracing). Setting this attribute on an input port has no effect on the analyze_graybox command. You need to exclude the scan_out ports of the core chains (or channel outputs if it's a design containing EDT hardware) from graybox analysis using this attribute because the tool has no way of knowing which outputs are core chain scan output ports. For more information about this attribute, refer to the analyze_graybox command description.
ijtag_c0	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_low_output_port_list attribute on the ICL module.
ijtag_c1	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_high_output_port_list attribute on the ICL module.
ijtag_connection_rule_option	An attribute that reflects the connection_rule_option attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_high_dft_signal_list	An attribute that reflects the forced_high_dft_signal_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_low_dft_signal_list	An attribute that reflects the forced_low_dft_signal_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.

ijtag_forced_high_output_port_list	An attribute that reflects the force_high_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_low_output_port_list	An attribute that reflects the force_low_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_function	String that reflects the port function of the ports as described in the ICL module. Its value, which is populated in the ICL extraction mode of the patterns -ijtag context, is the port function with an underscore separating the words. For example, a ScanInPort in ICL has icl_function equal to scan_in.
ijtag_function_modifier	An attribute that reflects the function_modifier attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_logical_hier_connection_from_src	An attribute that is set on output ports and contains the port on the module that is the logical source of the output port. It is extracted from the associated ICL module attribute ijtag_logical_connection = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_connection_from_src_inv	An attribute that is set on output ports and contains the port on the module that is the inverted logical source of the output port. It is extracted from the associated ICL module attribute ijtag_logical_connection_inv = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_connection_to_dst	This attribute is set on input ports and contains the ports on the module that are the logical destinations of the input port. It is extracted from the associated ICL module attribute ijtag_logical_connection = “{port1 port2} {port3 port4} ...”

ijtag_logical_hier_connection_to_dst_inv	This attribute is set on input ports and contains the ports on the module that are the inverted logical destinations of the input port. It is extracted from the associated ICL module attribute ijtag_logical_connection_inv = “{port1 port2} {port3 port4} ...”
index	Integer that specifies the index of the port that is already part of the name when the port is part of a bus.
is_bus	Boolean that specifies whether the port is part of a bus. This attribute is set to true when the port is part of a bus.
is_excluded_from_isolation_constraints	Boolean that specifies if the port should be excluded from CX constraints (input ports) or masking (output ports) when set to true. During core-level pattern generation, the tool automatically adds X constraints on most input ports and adds output masks on all output ports so that generated patterns can be mapped to the chip level for retargeting. Pulse-always clocks, pulse-in-capture clocks, constrained ports and internal primary inputs are excluded from these X constraints. You can exclude additional primary inputs from X constraints by setting this attribute to true; this is necessary if the ports are controlled by a Named Capture Procedure (NCP) that models chip-level constraints such as a clock controller located outside of the core. If a port is not a pulse-always or pulse-in-capture clock, is not constrained, and is excluded from isolation constraints (that is, the attribute is set to true on the port), the create_patterns command checks that the port is controlled (for example, by NCPs or clock control definitions). If the port is not controlled, the tool issues an R7 DRC violation. The default is false.
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
library_name	String that specifies the library the port belongs to.

loadboard_pad_decay	Specifies the leakage time of a pin. The attribute value must be a valid time value and time unit.
loadboard_pull_resistor	Specifies whether the port is a pullup or pulldown. Possible values are zero for pulldown, one for pullup, and an <EMPTY_STRING> for a normal port.
master_module_name	String that specifies the value of the master_name on the module object the port belongs to.
module_name	String that specifies the name of the module object the port belongs to.
name	String that specifies the name of the port object. One port object exists for each bit of a bused port and the name includes the index of the port. For example: A[3]
no_control_point	An attribute that specifies that no control point can be inserted on any top-level pin or a pin in any module when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.
no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.

no.observe_point	An attribute that specifies that no observe point can be inserted on any top-level pin or a pin in any module when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.
no.observe_reason	<p>String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no.observe_reason in the Module attribute section.</p>
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "port: <i>integer</i> "
object_type	Returns the object data type "port".
power_domain_name	Returns the name of the power domain of the current design. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain , refer to the <i>Tessent Glossary</i> .

preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins of the port will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
special_drive	String value extracted from the cell library data. See the read_cell_library command for information on how cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. Possible values are: open_drain open_source.
pull_drive	String value extracted from the cell library data. See the read_cell_library command for information on how cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. Possible values are: pull0 pull1.
static_dft_signal	A sting attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the static DFT signal.
tcd_bscan_function	A string attribute that reflects on the port the function the given port has within the matched TCD BoundaryScan description (see “ Tessent Core Description ” on page 3755). Only module which has the tcd_bscan_name attribute defined will have ports with the tcd_bscan_function defined. The property names found in the Interface wrapper gets transferred to the port they map to.

tcd_memory_function	A string attribute that reflects on the port the function the given port has within the matched TCD Memory description (see “ Tessent Core Description ” on page 3755). Only module which has the tcd_memory_name attribute defined will have ports with the tcd_memory_function defined. The function name found in the Core/Memory/Port wrapper gets transferred to the port they map to.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, Z, or “” (empty string). The default is the empty string which means the port is not tied to a value.

Inheritance

Built-in attribute inheritance from associated objects:

- None

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Port Specification

A port object (port_object) is an input, output, or inout interface of a module, which is specified by a string. Examples of port names can be “In1”, “ClkA”, and “Out3”.

Function Attribute Values

The following list shows the possible values for the function attribute. See the table in the [Pin Attributes](#) section of the *Tessent Cell Library Manual* for a complete list of available function values.

- active_high_clock
- active_high_reset
- active_high_set
- active_low_clock
- active_low_reset
- active_low_set
- asynch_disable
- asynch_disable_inv
- asynch_enable

- asynch_enable_inv
- clock_in
- clock_out
- data_in
- data_out
- data_out_inv
- func_enable
- func_enable_inv
- ground
- mux_in0
- mux_in1
- mux_in2
- mux_in3
- mux_in4
- mux_in5
- mux_in6
- mux_in7
- mux_out
- mux_select
- mux_select0
- mux_select1
- mux_select2
- negedge_clock
- posedge_clock
- power
- power_isolation_external
- power_isolation_internal
- scan_enable
- scan_enable_inv

- scan_in
- scan_out
- scan_out_inv
- scan_out_no_polarity
- tck
- tdi
- tdo
- test_enable
- test_enable_inv
- tie0
- tie1
- tms
- trst

Pad Function Attribute Values

The following list shows the possible values for the pad_function attribute. See the table in the [Pin Attributes](#) section of the *Tessent Cell Library Manual* for a complete list of available function values.

- to_pad
- enable_low
- enable_high
- pad_io
- pad_io_inv
- force_disable
- to_sje_mux_low
- to_sje_mux_high
- to_sji_mux
- to_sjo_mux
- select_jtag_enable
- select_jtag_in

- select_jtag_out
- init_data_dot6
- init_data_inv_dot6
- init_clock_dot6
- ac_mode_dot6
- from_pad
- from_sje_mux
- from_sji_mux
- from_sjo_mux
- test_data_dot6
- test_data_inv_dot6
- from_io
- from_io_inv
- to_io
- to_inv_io
- keep_tracing
- sample_pad
- input_enable_low
- input_enable_high
- init_posedge_clock_dot6
- init_negedge_clock_dot6
- init_enable_high_dot6
- init_enable_low_dot6
- parametric

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_fanins](#)
[get_fanouts](#)

```
get_common_parent_instance  
get_name_list  
report_attributes
```

Pin

A pin is an object on an instance that can be connected to other pins or ports with net objects. No pin object exists until the set_current_design command is executed.

The pin object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types.
- Port attributes are stored on the associated port of the pin and not on the pin object itself.
- Attributes defined on a net are automatically visible by a pin having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This behavior enables you to register an attribute for net object types and have the attribute fully visible from the root-level ports and the hierarchical pins.

Built-In Attributes

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
base_name	String that specifies the name of the pin without the leaf name.
bisr_power_domain_name	An attribute that can be set on BIST_SI pins of sub modules to associate a BISR chain to a power domain defined in the current level. You typically do not need to use this attribute if you have used consistent power domain name in your bottom-up flow but you can use this attribute on pins to map specific BISR chains on sub modules to power domain at the current level if they are different as documented in the bisr_segment_order_file section.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.

clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every pin found between the clock source and the memory and other clock in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.
created_by_synthesis	Boolean that indicates that the pin was created by synthesis. When a pin has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
direction	<p>String that specifies the direction of the pin. Possible values are as follows:</p> <ul style="list-style-type: none"> • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL)
dynamic_dft_signal	A sting attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the dynamic DFT signal.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_pin_id	Returns a gate_pin id when the pin has an associated gate_pin inside the flat model. Set to true for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the pin is of direction input and the pin is not connected, is connected to an undriven or tied net or is connected to a constant.
ijtag_logical_hier_connectio_n_from_src	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to its logical source.
ijtag_logical_hier_connectio_n_from_src_inv	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to its inverted logical source.
ijtag_logical_hier_connectio_n_to_dst	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to its logical destinations.
ijtag_logical_hier_connectio_n_to_dst_inv	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to its inverted logical destinations.

is_hard_module	Boolean that reports the state of the is_hard_module attribute on the instance on which the pin exists.
is_non_editable	Boolean that specifies that the pin is non-editable.
is_non_editable_reason	A string that specifies why the pin is non-editable as follows: <ul style="list-style-type: none"> • below_hard_module — The pin is on an instance that is inside an instance that has is_hard_module equal to 1 • created_by_quick_synthesis — The pin is on an instance that was created by quick synthesis • below_matched_icl_module — The pin is on an instance that is below a module that has a matched ICL module
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the pin (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_leaf_name attribute.
name	String that specifies the hierarchical name of the pin object relative to the current design. One pin object exists for each bit of a bused port; the name includes the index of the port appended to its associated instance name. For example: u1/u2/A[3]. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_name attribute.
no_control_point	An attribute that specifies that no control point can be inserted on a pin in a specific instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.
no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.
no_observe_point	An attribute that specifies that no observe point can be inserted on a pin in a specific instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.

no.observe.reason	<p>String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no.observe.reason in the Module attribute section.</p>
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "pin:integer"
object_type	Returns the object data type “pin”.
parent_instance	<p>String that specifies the instance name of the instance on which the pin is instantiated. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the <leaf_instance_name>/<leaf_pin_name> part of the path.</p> <p>For example:</p> <p>parent_instance of pin (u2/u7/a) is u2 parent_instance of pin (u2/b) is null</p>
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute on the pin’s parent instance is set to true. This attribute is set to true if the is_hard_module attribute on the pin’s parent instance is set to true. The default is false.
power_domain_name	Returns the name of the power domain of the instance to which this pin belongs. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. For a definition of Power Domain , refer to the <i>Tessent Glossary</i> .
pre_synthesis_name	<p>String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name will only ever be different from the name when the pin has one or more generate block in its ancestry that the tool synthesized during quick synthesis.</p>

preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the hierarchical pin will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin will be visible in the flat model. If you enable this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the static DFT signal.
tie_value	String that indicates the value to which the pin is tied. Possible values are: 0, 1, X, Z, or "" (empty string). The default is the empty string which means the pin is not tied to a value.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.

Inheritance

Built-in attribute inheritance from associated port objects:

- bus_left_index
- bus_name
- bus_range_direction
- bus_right_index
- bus_width
- direction
- function
- ijtag_function
- index
- is_bus

- library_name
- master_module_name
- module_name
- preserve_boundary
- special_drive
- pull_drive

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Pin Specification

A pin object (pin_object) is specified by a complete hierarchical pin pathname including the pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Related Topics

- [get_attribute_list](#)
- [get_attribute_value_list](#)
- [get_fanins](#)
- [get_fanouts](#)
- [get_name_list](#)
- [get_pins](#)
- [report_attributes](#)

Net

For every port and pin object, a net object with the same name exists. No net objects exist until the set_current_design command is executed.

When objects are referenced by name in the editing and attribute commands, if a specified name exists as both a pin and a net or a port and a net, the pin or port is selected. If you want to force the selection of the net object that has the same name as a pin or port, use the [get_nets](#) introspection command. For examples of this usage, see the examples shown on the [move_connections](#) command reference page.

Attributes defined on a net are automatically visible by a pin or port having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin or port are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This is true for attributes on pins inherited from their associated port. This behavior enables you to register an attribute for the net object type

and have the attribute fully visible from the hierarchical pins and from the ports of the current design.

Built-In Attributes

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every net found between the clock source and the memory and other clock in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.
created_by_synthesis	Boolean that indicates that the net was created by synthesis. When a net has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
dynamic_dft_signal	A sting attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the dynamic DFT signal.

has_functional_source	Returns true unless the net is undriven or tied.
has_rtl	Boolean that indicates whether the net is connected to an RTL construct, such as an always block. Also indicates when the driver of the net is an RTL construct. The attribute is true under the following circumstances: <ul style="list-style-type: none">• The net is used in an RTL expression.• The net is directly connected to a complex pin (not via an assign). The attribute is always false in the -no_rtl flow.
has_rtl_fanin	Boolean that indicates whether the net is connected to a driver embedded inside an RTL construct, such as an always block.
has_rtl_fanout	Boolean that indicates whether the net is connected to one or more sinks embedded inside an RTL construct, such as an always block.
index	Integer that specifies the index of the net that is already part of the name when the net is part of a bus.
in_graybox	Set to true when the net is included in the graybox netlist. Set to false when the net is excluded from the graybox netlist, which is the default for all the nets in the design. The attribute value can be changed by the analyze_graybox or set_attribute_value command. For more information about this attribute, refer to the analyze_graybox command description.
is_bus	Boolean that specifies whether the net is a part of a bus. This attribute is set to true when the net is part of a bus.
is_non_editable	Boolean that specifies that the net is non-editable.
is_non_editable_reason	A string that specifies why the net is non-editable as follows: <ul style="list-style-type: none">• below_hard_module — The net is inside an instance that has is_hard_module equal to 1• created_by_quick_synthesis — The net was created by quick synthesis• below_matched_icl_module — The net is below a module that has a matched ICL module
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_block_id	The leaf_name of nets located inside generate blocks have the generate block ids joined by periods prepended to their local name. This attribute returns the block_id part of the leaf_name. It is null for nets that are not inside generate blocks.
leaf_name	String that specifies the name of the net (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_leaf_name attribute.

leaf_name_hash	<p>This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2.</p> <p>The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with <i>parent_module.leaf_name_hash.pin_leaf_name</i> entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage. Note that the value of this attribute always refer to the pre_synthesis_name just like the pre_synthesis_leaf_name attribute.</p>
library_name	String that specifies the library of the module in which the leaf net resides.
master_module_name	String that specifies the value of the master_name in which the leaf net resides.
module_name	String that specifies the name of the module object in which the leaf net resides.
name	String that specifies the hierarchical name of the net object relative to current design. One net object exists for each bit of a bused net; the name includes the index of the net appended to its associated instance name. For example: u1/u2/A[3]. After leaving insertion mode in the tool, this attribute will show the value of the unrolled_name attribute.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "net:integer"
object_type	Returns the object data type "net".
parent_instance	String that specifies the instance name in which the net exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
parent_is_hard_module	Boolean that specifies whether the module containing the net has the is_hard_module attribute set to true. This attribute is set to true if the module containing the net has the is_hard_module attribute set to true. The default is false.

pre_synthesis_leaf_name	String that specifies the original RTL leaf name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the leaf_name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_leaf_name will only ever be different from the leaf_name when the net is inside a generate block that the tool synthesized during quick synthesis.
pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name will only ever be different from the name when the net has one or more generate block in its ancestry that the tool synthesized during quick synthesis.
power_domain_name	Returns the name of the power domain of the instance in which this net exists. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the net will have an inferred buffer created in the flat model to preserve the net name. If you set this attribute on a net of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
specified_internal_scan_only	An attribute used to specify that the flip-flop instance inferred from the given net during quick synthesis is to remain internal during wrapper analysis. Dedicated isolation wrapper cells will be inferred to isolate the input and output pins of the instances if any interacts with the primary input and output ports.
static_dft_signal	A sting attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the static DFT signal.
tie_value	String that indicates the value to which the net is tied. Possible values are: 0, 1, X, Z, or "" (empty string). The default is the empty string which means the net is not tied to a value.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.

unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute will show the name the object will have when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.
---------------	---

Inheritance

Built-in attribute inheritance from associated pin objects:

- Attributes defined on pins or ports having the same name as the net, if the attribute is not also registered on the net object type.

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Net Specification

A net object (net_object), sometimes called just a net, is a wire that connects the pins of instances in the design. A net is specified by a hierarchical net pathname. In the following example, u5/u2/Q is a pin that drives the net u5/u2_Q. This example is based on the example design presented in the “[Hierarchical Design Data Model](#)” section of the *Tessent Shell User’s Manual*.

```
SETUP> get_fanouts u5/u2/Q
{u5/u1/A0 u5/u3/A0}

SETUP> get_fanouts u5/u2/Q -stop_on net
{u5/u2_Q}
```

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_fanins](#)
[get_fanouts](#)
[get_name_list](#)
[get_nets](#)
[report_attributes](#)

Pseudo_port

A pseudo_port represents a user-added primary input or primary output.

A pseudo_port object enables the introspection of user-added primary inputs and outputs such as those created using the commands [add_primary_inputs](#), [add_primary_outputs](#), and [add_clocks](#).

A pseudo_port is not a real port in that it is not present in the original design, it cannot be used for design editing, and it is not reflected in designs written out using the write_design command. However, the pseudo_port is reflected in the flat model and it is used for analysis operations that use the flat model such as ATPG and simulation.

A pseudo_port shares some of the same attributes as the port object. Pseudo ports have a simple non-hierarchical name which may be the same as the pin's hierarchical name (for non-grouped pseudo_ports).

Pseudo_ports can be introspected with the [get_ports](#) command. The “real” pins that underlie a pseudo_port can be introspected with the “[get_pins -of_pseudo_ports pseudo_port_objects](#)” command.

The pseudo_port object type does not support attribute inheritance to the real underlying objects (pins/nets) associated to it. That is, the real design object behind the pseudo_port does not see the attributes set on this type of port.

Built-In Attributes

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every pseudo_port on which a clock was defined. The string contains the label associated to the clock defined on the pseudo_port. It is also set by DFT_C6 on every node between the clock and the sequential elements.
constraint_value	String that specifies that the port’s primary input was set to a constant with the add_input_constraints command. When set to C0, CT0, C1, CT1, CX, or Cz, this attribute displays 0, 0, 1, 1, x, or z, respectively. The default is null.
constraint_value_during_load_unload	String that specifies a constant value for a top-level pseudo_port during load_unload test procedure retargeting. Possible values are: 0, 1, X, and Z. Constraining the value of the top-level pseudo_port enables tracing of signals from the core level to the chip level.

direction	String that specifies the direction of the pseudo_port. Possible values are as follows: <ul style="list-style-type: none">• input• output
gate_pin_id	Returns the id of the gate_pin associated with the pseudo_port inside the flat model.
is_bus	Boolean that specifies whether the pseudo_port is part of a bus. This attribute is always set to false. The pseudo_port object type does not support busses.
is_valid	Boolean that specifies whether the object still exists.
name	String that specifies the name of the pseudo_port object.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "pseudo_port: integer"
object_type	Returns the object data type "pseudo_port".

Inheritance

Built-in attribute inheritance from associated objects:

- None

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Related Topics

[add_clocks](#)
[add_primary_inputs](#)
[add_primary_outputs](#)
[get_attribute_list](#)
[get_attribute_value_list](#)
[get_fanins](#)
[get_fanouts](#)
[get_name_list](#)
[get_ports](#)
[report_attributes](#)

Flat Design Data Model

The flat design data model is an internal, flattened representation of the hierarchical design that is produced when exiting setup mode.

You can also explicitly create the flat model using the `create_flat_model` command.

The flat design data model consists of gates that are connected together. A gate is an instance of a primitive module, and a `gate_pin` object represents a pin on a gate instance. Gates and `gate_pins` have no unique instance name, instead having a unique ID that is available to differentiate one from another. The format of the ID is two integers separated by a period character. The first integer represents the gate and the second integer represents the `gate_pin`. However, this ID does not remain in place from one netlist version to the next or from one Tesson Shell invocation to the next. For that reason, you should avoid hardcoding a gate ID into a script.

Flat Design Object Types	3108
<code>Gate_pin</code>	3108

Flat Design Object Types

The `gate_pin` is the only object type in the flat model.

Gate_pin 3108

Gate_pin

A `gate_pin` represents a pin on a gate instance in a flattened design.

Built-In Attributes

The `gate_pin` object type has the following attributes:

active_clock_edge	A string attribute having the value “trailing” or “leading” which is set on the <code>gate_pins</code> of the DFF primitives. The attribute is set on the CK input <code>gate_pin</code> and its associated D input <code>gate_pin</code> . The attribute is also visible on the Q <code>gate_pin</code> . For multi clock DFF primitive, the value seen on the Q <code>gate_pin</code> is the list of all values present on all CK <code>gate_pins</code> . The value of the attribute is “leading” when the clock source has an off state of 0 and there is no inversion between the source and the DFF CK pin or when the clock source has an off state of 1 and there is an inversion between the source and the DFF CK pin. In the other two situations, the value is “trailing”.
active_polarity	A string attribute having the value “on_state” or “off_state” which is set on the <code>gate_pins</code> of the DLAT primitives. The attribute is set on the CK input <code>gate_pin</code> and its associated D input <code>gate_pin</code> . The attribute is also visible on the Q <code>gate_pin</code> . For multi clock DLAT primitive, the value seen on the Q <code>gate_pin</code> is the list of all values present on all CK <code>gate_pins</code> . The value of the attribute is “on_state” when the latch is open when the clock source is set to it its on state. The value of the attribute is “off_state” when the latch is open when the clock source is set to it its off state.
base_name	String that specifies the name of the instance in which the primitive module of the <code>gate_pin</code> is instantiated.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every <code>gate_pin</code> found between the clock source and the memory and other clocks in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.

default_fault_classification_0	<p>String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-0 or slow-to-rise. You use the “set_attribute_value” command to set the desired default fault classification. During the flow, when the “add_faults” and “load_faults” commands are called, or when the tool internally adds faults for scan-based test pattern generation, the fault sites marked with this attribute will be initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it will be ignored.</p> <p>This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test.</p> <p>See Example 4 in the set_attributes_value command for a typical use model</p>
default_fault_classification_1	<p>String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-1 or slow-to-fall. You use the “set_attribute_value” command to set the desired default fault classification. During the flow, when the “add_faults” and “load_faults” commands are called, or when the tool internally adds faults for scan-based test generation, the fault sites marked with this attribute will be initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it will be ignored.</p> <p>This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test.</p> <p>See Example 4 in the set_attributes_value command for a typical use model</p>

direction	<p>String that specifies the direction of the gate_pin object.</p> <p>Possible values are as follows:</p> <ul style="list-style-type: none"> • input • output
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the dynamic DFT signal.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_id	Integer that specifies the unique ID of the gate to which the gate_pin belongs. This integer is identical to the first integer in the gate_pin's id attribute. This value is not persistent and changes every time you create a flat model.
gate_input_count	Integer that specifies the number of inputs on the gate associated with gate_pin. Use this attribute to filter a collection of gate_pins and quickly separate single input gates from multi-input gates. For example, to find all multi-input gates along the clock tree: <code>get_gate_pins -filter "clock_domain_label && direction == output && primitive_name != DFF && primitive_name != DLAT && gate_input_count > 1"</code>
hier_pin_name	String that specifies a hierarchical pin name when the gate_pin is associated to a hierarchical pin. This attribute is null when the gate pin is not associated to a hierarchical pin. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.
id	Specifies a unique ID for each gate_pin that consists of two integers separated by a period: <i>gate_id.pin_index</i>
is_internal_scan_only	Boolean that specifies whether the gate_pin object is internal scan only.
is_non_scannable	Boolean that specifies that the gate_pin object is non-scannable. Default is false.
is_valid	Boolean that specifies whether the gate_pin object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.

leaf_name	String that specifies the hierarchical pin name without the hierarchical path when the gate_pin is associated to a hierarchical pin. This is null by default.
name	String that specifies name of the gate_pin object. The value is the same as the hier_pin_name attribute when not empty; otherwise, the value is the same as the id attribute of the gate_pin object. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.
no_control_point	An attribute that specifies that no control point can be inserted on a gate_pin in a specific instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.
no_control_reason	<p>String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.</p>
no_observe_point	An attribute that specifies that no observe point can be inserted on a gate_pin in a specific instance when the attribute value is set to true. Possible values are true, false and unspecified. The default value is unspecified which means that control points will be allowed unless the attribute is set to false in one of its parent modules or instances.
no_observe_reason	<p>String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified.</p> <p>The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode.</p> <p>For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.</p>
object_type	Returns the object data type “gate_pin”.

parent_instance	If the gate_pin is associated to a hierarchical pin, this string specifies the name of the instance on which the hierarchical pin is instantiated. If the gate_pin isn't associated to any hierarchical pin, this attribute specifies the name of the instance on which the primitive module is instantiated.
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute is set to true on the parent module of the gate_pin. This attribute is false if the parent module of the gate_pin does not have the is_hard_module attribute set to true. This attribute is true for primitives instantiated directly into a design module which does not have is_hard_module set to true. This attribute can also be true for a gate_pin that has the hier_pin_name pointing to a pin on a module instantiated into a non-hard design module.
pin_index	Integer that specifies the <i>id</i> of the pin. This integer is identical to the second integer of the gate_pin's <i>id</i> attribute. The pin index is persistent. The pin_index is the index of the gate_pin on the primitive module, where 0 is the index of the single output and integers larger than 0 correspond to the inputs.
primitive_name	Name of the primitive module associated with the gate on which the gate_pin exists. The primitive_name attribute values for all primitive types are listed in Table 8-1 .
primitive_port_name	Name of the port on the primitive module. The primitive_port_name attribute values for all primitive types are listed in Table 8-1 .
power_domain_name	Returns the name of the power domain in which the gate (instance) to which this gate_pin is attached exists. This attribute is inherited from the instance.
simulation_force_value	String that contains the simulation force value that is specified for a gate_pin with the add_simulation_forces command.
simulation_value	String that contains the simulation value on a gate_pin.
static_dft_signal	A sting attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the static DFT signal.

Built-In Primitives and Primitive Port Names

Table 8-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
BUF	Buffer	I	O
INV	Inverter	I	O
AND	AND gate	I0, I1, ..., In	O
NAND	Negated AND	I0, I1, ..., In	O
OR	OR gate	I0, I1, ..., In	O
NOR	Negated OR gate	I0, I1, ..., In	O
XOR	Exclusive OR	I0, I1, ..., In	O
NXOR	Negated Exclusive OR	I0, I1, ..., In	O
DFF	D Flip-flop	S, R, CK (CK0, CK1, ... CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
DLAT	D latch	S, R, CK (CK0, CK1, ... CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
PI	Primary input	-	O
PO	Primary output	I	-
TIE0	Tie-0	I	O
TIE1	Tie-1	I	O
TIEX	Tie-X	I	O
TSD	Tri-state driver with active high at control input.	EN, D	O

Table 8-1. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
BUS	Bus primitive. Drivers must include at least one TSH or switch gate.	I0, I1, ..., In	O
ZVAL	Z converter gate, converts Z to X: dummy gate inserted between output of the gate taking Z and the binary gate input.	I	O
WIRE	Wire	I0, I1, ..., In	O
MUX	Multiplexer	S, D0, D1	O
TIEZ	Tie-Z	I0, I1, ..., In	O
NMOS	Switch gate	G (gate), S (source)	D (drain)
ZHOLD	Z-hold allows modeling of tri-state nets such that they can retain the previous state when the net goes to a Z value.	I	O
PBUS	Pull-bus gate. Used when combining strong (ZW) bus and weak bus signals (e.g. TIE0) together. The strong value (input 1) always goes to the output, unless the value is Z, in which case the weak value (input 2) propagates to the output.	I0, I1	O

Table 8-1. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
WBUS	Weak bus primitive	I0, I1, ..., In	O
BUFZ	Buffer allowing Z value at input.	I	O
XDET	X-detector gate: input 'X' results in output value '1'; otherwise output is '0'.	I	O
ZDET	Z-detector gate: input 'Z' results in output value '1'; otherwise output is '0'.	I	O
RAM	Read-write memory	S, R, WCK (WCK0, WCK1, ... WCKw in case of multiple write ports), WEN (WEN0, WEN1, ... WENw in case of multiple write ports), WADDR[0...i] (WADDR0[0...i], WADDR1[0...i], ... WADDRw[0...i] in case of multiple write ports), WDIN[0...j] (WDIN0[0...j], WDIN1[0...j], ... WDIN w[0...j] in case of multiple write ports), RCK, REN, RADDR[0...i] (RADDR0[0...i], RADDR1[0...i], ... RADDRr[0...i] in case of multiple read ports)	Q
ROM	Read-only memory	CK, EN, ADDR[0...i] (ADDR0[0...i], ADDR1[0...i], ... ADDRr[0...i] in case of multiple read ports)	Q

Table 8-1. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
RAM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, ..., In	O
ROM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, ..., In	O
FB_BUF	Feedback buffer: internal gate type to break combinational loops in the design.	I	O
PGEN	Pulse generator: circuitry that creates a pulse at its output when active.	I	O
CORE	Core (obsolete)	I0, I1, ..., In	O
CORE_OUT	Output of core, place holder for multiple core outputs in the flat model (obsolete).	I0, I1, ..., In	O

Inheritance

Built-in inherited attributes:

- During flattening, attributes defined on hierarchical pins (cell or design) are copied to the gate_pin objects when there is a gate_pin associated to the hierarchical pin. Those are fault sites on cell pins, and those pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Related Commands

get_attribute_list	get_gate_pins
get_attribute_value_list	get_pins
report_attributes	get_name_list

Gate_pin Specification

A gate_pin object (gate_pin_object) is assigned a number ID rather than having an actual name. The number ID is a unique gate pin object id of the format “gate_id.gate_pin_id”. For example an output of an AND gate in the flattened design might have an ID of “1325.0”.

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[report_attributes](#)
[get_gate_pins](#)
[get_pins](#)
[get_name_list](#)

ICL Data Model

The ICL data model defines objects as one of the following four data types: `icl_module`, `icl_instance`, `icl_port`, and `icl_pin`. The `icl_module` and the `icl_port` object types are created as soon as ICL modules are read in using the `read_icl` command.

Each `icl_module` object has its list of `icl_port` objects. Once the current design is set using the `set_current_design` command, the ICL data model is elaborated downward from the current design and `icl_instance` objects are created for each instance of `icl_module`, and `icl_pin` objects are created for each port of the modules associated to the `icl_instances`. The `icl_instance` and `icl_pin` objects are hierarchical objects and therefore only exist after the current design is set.

The purpose of the Instrument Connectivity Language (ICL) is to describe the elements that comprise the IJTAG network as well as their logical (though not necessarily physical) connections to each other and to the instruments at the endpoints of the network. ICL bears a loose resemblance to a hierarchical netlist such as Verilog; it is organized by modules that may contain instances of other modules, and it describes the connections between the pins of the instances. However, it is important to note that ICL is not a complete netlist; connections are port-to-port rather than through nets, and ICL freely uses abstraction in order to omit the detailed physical construction of the circuitry — only the behavioral operation of the network is represented.

Beyond merely documenting the structure of the network, the purpose of ICL is to enable an operation called “test pattern retargeting.” With test pattern retargeting, the sequences of test patterns, written in PDL, that are to be applied at the instrument boundaries are mapped through the IJTAG network to a higher level of hierarchy and ultimately to chip level pins. Tesson Shell reads and stores ICL information in a data structure and uses that information to retarget the instrument-level test patterns into test patterns that can then be applied at a higher level.

If a Verilog or VHDL representation has also been read prior to issuing the `set_current_design` command, the port list on the design and the ICL module corresponding to the current design is compared to make sure that all ports declared in the ICL module also exist on the design module. The design module is allowed to have more ports than the ICL module. In fact, it is very likely the design module will have more ports because the ICL module only include the ports with an IJTAG function while the design module includes all ports on the physical module. The test bench created for the current design uses the complete port list read in from the design view when it is supplied.

ICL Object Types.....	3119
<code>icl_module</code>	3119
<code>icl_port</code>	3126
<code>icl_instance</code>	3139
<code>icl_pin</code>	3141

ICL Object Types

The `icl_module`, `icl_instance`, `icl_port`, and `icl_pin` object types are described on the following reference pages.

<code>icl_module</code>	3119
<code>icl_port</code>	3126
<code>icl_instance</code>	3139
<code>icl_pin</code>	3141

icl_module

One module is designated as the target module using the `set_current_design` command. Instances and pins are hierarchical objects defined relative to the current module.

The following properties apply to the `icl_module` object type:

- An `icl_module` object has a simple non-hierarchical name. Only one `icl_module` with a given name is allowed within a given namespace. Currently there is only one global namespace for ICL in Tessent Shell.
- An `icl_module` can be parameterized and its parameters can be overridden during instantiation. For each combination of parameter overrides, an `icl_module` object corresponding to the parameterized view exists. The name of the parameterized view is unqualified with a `@PV#` suffix where # is a unique integer starting from 1 and incrementing per unique parameterized view.
- One master `icl_module` exists for each module definition in the input files. The master `icl_module` is the one that contains the ICL module definition with default parameter values.
- Attributes defined on `icl_modules` are automatically visible as read-only attributes on the instance objects when the same attribute name is not also registered on the `icl_instance` object type.
- Attributes defined with the `set_attribute_options -applies_to_child_instances` switch set to On automatically become visible to all child instances of each instance of the `icl_module`.
- `icl_module` attributes are stored on all parameterized views of an `icl_module`. When an attribute value is set on an `icl_module` object, the attribute value is set on all `icl_module` objects having the same `master_name` within a name space.

For more information on attributes, see “[Attributes](#)” on page 3159.

Built-In Attributes

The `icl_module` object type has the following attributes:

<code>bsdl_file</code>	An attribute that is attached to the ICL module by Tessent Shell, if the ICL module contains an AccessLink statement and the <code>-bsdl_file</code> option of the <code>read_icl</code> command has been used. The presence of the AccessLink statement in the ICL module instructs the tool to create ICL for the TAP controller and to integrate it into the IJTAG network. The BSDL file is required for the details of the TAP controller, such as instruction names and opcodes. The attribute <code>bsdl_file</code> holds the information about the BSDL file that provided those TAP controller specifications.
<code>file_path_name</code>	String that specifies the pathname to the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been saved and read back in.
<code>forced_high_input_port_list</code>	Lists all input ports that were constrained to 1 during ICL extraction. Those constraints are automatically re-applied in the generated patterns. They are also verified to be properly simulated when running ICL extract at the next level. An I5 DRC violation is generated if the simulated value on any of those pins is not 1.
<code>forced_high_internal_input_port_list</code>	Lists all internal pins that were constrained to 1 during ICL extraction. Those constraints are reported in this attribute for debugging and serve no other purpose.
<code>forced_high_output_port_list</code>	Lists output ports of IJTAG instruments that must be forced high during ICL extraction in order to see all the connections between the instruments. You must use this attribute with care and typically use it for logictest testmode signals sourced by the ICL network itself. For example, if you have a TDR bit driving a TM signal and this TM signal puts the circuit in scan mode, you may need to force this signal to its inactive value for the rest of the ICL network to be traceable.
<code>forced_low_input_port_list</code>	Lists all input ports that were constrained to 0 during ICL extraction. Those constraints are automatically re-applied in the generated patterns. They are also verified to be properly simulated when running ICL extract at the next level. An I5 DRC violation is generated if the simulated value on any of those pins is not 0.
<code>forced_low_internal_input_port_list</code>	Lists all internal pins that were constrained to 0 during ICL extraction. Those constraints are reported in this attribute for debugging and serve no other purpose.

forced_low_output_port_list	List output ports of IJTAG instruments that must be forced low during ICL extraction in order to see all the connections between the instruments. You must use this attribute with care and typically use it for logictest testmode signals sourced by the ICL network itself. For example, if you have a TDR bit driving a TM signal and this TM signal puts the circuit in scan mode, you may need to force this signal to its inactive value for the rest of the ICL network to be traceable.
icl_extraction_date	Reports the date and time when the ICL module was extracted. It is defined in the ICL module and created by ICL extraction.
ijtag_logical_connection	An attribute that creates a logical connection between source and destination ports. These attributes are transferred to the <code>ijtag_logical_hier_connection_from_src</code> and <code>ijtag_logical_hier_connection_to_dst</code> attributes on the matching module object. ICL extraction traces through those connections as if there was a buffer between the two ports. The syntax is <code>attribute ijtag_logical_connection = "{src1 dst1} {src2 dst2} ...";</code> The source and destination ports may or may not be declared as ports in the ICL module but they must be declared with the proper direction in the design module object. To create a logical connection that is not between an input and output port of a module having an associated ICL module, use the <code>add_ijtag_logical_connection</code> command.
ijtag_logical_connection_inv	An attribute that creates an inverted logical connection between source and destination ports. These attributes are transferred to the <code>ijtag_logical_hier_connection_from_src_inv</code> and <code>ijtag_logical_hier_connection_to_dst_inv</code> attributes on the matching module object. ICL extraction traces through those connections as if there was a buffer between the two ports. The syntax is <code>attribute ijtag_logical_connection_inv = "{src1 dst1} {src2 dst2} ...";</code> The source and destination ports may or may not be declared as ports in the ICL module but they must be declared with the proper direction in the design module object. To create a logical connection that is not between an input and output port of a module having an associated ICL module, use the <code>add_ijtag_logical_connection</code> command.
is_created	Boolean that specifies whether the module was created by ICL extraction. This attribute is set to true for modules created by ICL extraction. This attribute is not exported to the ICL file. Once the ICL module is re-read, this attribute is false.

is_modified	Boolean that specifies whether an ICL module contains attributes that have been added or modified after the module was read in.
is_valid	Boolean that specifies whether the ICL module is deleted or still valid. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
keep_active_during_scan_test	Boolean that specifies that the given ICL module is a node that must remain active during scan test so as not to be disturbed and be usable for scan test_setup. See the create_dft_specification command description for more information on how it is used. The default is “true”.
line_number	Integer that specifies a line number in the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been saved and read back in.
master_name	String that specifies the name of the icl_module without the optional parameterized view suffix.
name	String that specifies the name of the icl_module with an optional parameterized view suffix @PV#. ICL modules are defined from files read into the ICL database with the read_icl command. For example, the name of a parameterized view of the master icl_module ModA is: ModA@PV1
object_type	Returns the object data type “icl_module”.
partially_forced_input_port_list	Lists all bus ports with at least one element in the forced_low_input_port_list and forced_high_input_port_list. This attribute is useful when creating the testbench using the ICL port list in order to get the complete bus range of the constrained bused ports. This attribute is not used when the Verilog or VHDL design module is read before setting the current design in the patterns -ijtag context because the top-level port list is then extracted from the design module.
scan_interface_list	Lists all ScanInterface names defined within the ICL module.
tessent_boundary_scan_reg	A string attribute that identifies a ScanRegister inside the ICL module as the BOUNDARY register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “ BoundaryScan ” on page 3593 for more information.
tessent_bypass_reg	A string attribute that identifies a ScanRegister inside the ICL module as the BYPASS register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “ BoundaryScan ” on page 3593 for more information.

tessent_design_id	A string attribute that is set on the current ICL module when it is created during ICL extraction to reflect the design_identifier associated with the current design. See the “ set_context -design_identifier ” command and switch for more information about this identifier.
tessent_design_level	A string attribute that is set on the current ICL module when it is created during ICL extraction to reflect the design_level associated with the current design. See the set_design_level command for more information about this value.
tessent_device_id_reg	A string attribute that identifies a ScanRegister inside the ICL module as the DEVID register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “ BoundaryScan ” on page 3593 for more information.
tessent_dft_function	A string attribute used to identify an ICL module having special meaning during DFT. For example, the create_dft_specification command uses this attribute to identify the Sib(sti) and Sib(sri) so that they can be found again during the next insertion pass.
tessent_ignore_during_icl_verification	A string attribute used to identify an ICL module that is to be automatically excluded from the ICLNetworkVerify patterns or from the patterns created using the create_icl_verification_patterns command.
tessent_ignored_design_instances	A string attribute reported by ICL extract to reflect the list of design instances ignored during ICL extraction. These are the design instances with an ICL module for which the Boolean attribute ignore_during_icl_extraction has been set. The format of this attribute is a list of 3-element lists as shown here: { {<IclModuleName>} {<DesignModuleName>} {<DesignInstancePath>} ... }
tessent_instruction_reg	A string attribute that identifies a ScanRegister inside the ICL module as the INSTRUCTION register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “ BoundaryScan ” on page 3593 for more information.
tessent_instrument_container	String that specifies the base name of the directory that contains the Tessent Instrument Description (TID) file and any other IP-specific data. This attribute is required for any ICL module that has IP-specific data.
tessent_instrument_subtype	String that identifies the sub-type of the ICL module for one of the known instrument types. The value of this attribute is only used when the tessent_instrument_type attribute is also set.

tessent_instrument_type	String that identifies the ICL module as one of the known instrument types within the tool. When this attribute is set, the tessent_signature attribute must also be set with a valid signature string.
tessent_is_physical_module	Boolean that is set on the top ICL module when the extract_icl command runs and the level was set to chip or physical_block using the set_design_level command.
tessent_repair_word_size	Used for physical blocks with BISR chains. This value reports the maximal repair word size used in the block. The repair word size is the number of bits read from the fuse box. This value is used to generate the compression hardware in the fuse box controller.
tessent rtl pre trace stop port_list	Lists the names of ports on which the tool stops the ICL tracing during the ICL extraction pre-trace phase. This is useful to prevent the unnecessary synthesis of module during ICL extraction.
tessent_signature	A string that is used to identify that an ICL module has not been modified after its creation. An ICL module can be read in without a tessent_signature or with a valid tessent_signature but a module with an invalid signature generates an error and is rejected during the read_icl command.
test_setup_procfile	Reports the filename of the procfile that contained the optional test_setup procedure used during ICL extraction. It currently is only informational to remind the user to reuse the same test_setup file during IJTAG pattern retargeting.
tessent_use_in_dft_specification	String that you can set on an ICL module in its ICL file to force the create_dft_specification command to ignore instances of this module. The legal value is auto or false.
use_in_icl_extraction	<p>String that specifies if the instances of the corresponding design modules are unconditionally used in ICL Extraction or if they are used only on-demand (corresponding design modules: design modules that have been mapped to the ICL module which owns this attribute)</p> <p>The attribute can have the following values:</p> <ul style="list-style-type: none"> • always — All instances of the corresponding design modules are used as starting points for ICL Extraction (default) • when_reached_by_other_modules — An instance of a corresponding design module is used in ICL Extraction only if it has been reached by tracing from another instance or top module port.

Related Commands

get_attribute_list	get_icl_scan_interface_port_list
------------------------------------	--

get_attribute_value_list	get_iproc_argument_default
get_icl_fanins	get_iproc_argument_list
get_icl_fanouts	get_iproc_body
get_icl_instances	get_iproc_list
get_icl_modules	get_name_list
get_icl_module_parameter_list	report_attributes
get_icl_module_parameter_value	report_icl_modules
get_icl_ports	report_module_matching
get_icl_scan_interface_list	report_module_matching_options

ICL Module Specification

An ICL module object is specified by an ICL module name which is a string. For example: “modA”.

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_icl_fanins](#)
[get_icl_fanouts](#)
[get_icl_instances](#)
[get_icl_modules](#)
[get_icl_module_parameter_list](#)
[get_icl_module_parameter_value](#)
[get_icl_ports](#)
[get_icl_scan_interface_list](#)
[get_iproc_argument_default](#)
[get_iproc_argument_list](#)
[get_iproc_body](#)
[get_iproc_list](#)
[get_name_list](#)
[report_attributes](#)
[report_icl_modules](#)
[report_module_matching](#)

[report_module_matching_options](#)

icl_port

Several properties apply to the icl_port object type.

- An icl_port object has a simple non-hierarchical name.
- An icl_port is an object on an icl_module and can have an input or output direction.
- One port object exists for each bit of a bussed port so every bit of a bussed port can be attributed independently.
- Attributes defined on icl_port objects are automatically visible as read-only attributes on the associated [icl_pin](#) objects unless the attribute is also registered for the [icl_pin](#) object type.
- Attributes set on an icl_port of one icl_module are set on the same port of each icl_module having a common master_name. Because different parameterized views can have more or less icl_port objects when the size of a bussed icl_port is parameterized, attributes on the other icl_module having the common master_name are only copies if the port object actually exists on the given parameterized view.

For more information on attributes, see “[Attributes](#)” on page 3159.

Built-In Attributes

The icl_port object type has the following attributes

active_polarity	An integer attribute defined on the ResetPort and ToResetPort ICL port objects which reflects the value of the ActivePolarity property in the ICL definition.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.

compliance_value	<p>This attribute specifies the compliance value of the ICL port and as such represents the COMPLIANCE_PATTERNS specification of an IEEE 1149.1 BSDL file. The attribute value can be specified as a sized or unsized binary number using legal ICL digits 0, 1 and X. If the attribute value is specified as a sized number, the size must match the size of the object. If the attribute value is specified as an unsized number, it is padded with 0s or Xs if necessary, dependent on the leftmost specified bit. The right-most bit (the least significant bit) of the attribute value represents the right-most bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on.</p> <p>This attribute is set automatically on all compliance enable ports, if the ICL top module contains an AccessLink statement of type STD_1149_1_2001 or STD_1149_1_2013. If the corresponding BSDL file contains more than one compliance pattern, the first compliance pattern is used to derive the attribute values.</p> <p>The compliance_value attribute determines the values that are applied to the ICL ports at the beginning of an IJTAG pattern set, see open_pattern_set.</p>
------------------	--

connection_rule_option	<p>Specified inside the Port wrappers of the ICL modules and used to modify the connection rules for the corresponding port. You use one of the following values to achieve the described behavior.</p> <ul style="list-style-type: none">• allowed_no_source — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations.• allowed_tied — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high or low. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations.• allowed_tied_low — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied low. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations.• allowed_tied_high — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations.• allowed_no_destination — Allowed on output ports of an ICL instrument to indicate that this output port is allowed not to be connected to a valid destination. You typically use this for DataOutPort and ScanOutPorts of host ports which are not always used in all instantiations.• auxiliary_data — Identifies the DataOutPort as an active-high auxiliary data pin. When tracing from a port with connection rule option “auxiliary_data”, ICL Extraction asserts all ports on all instances declared as an auxiliary_data or auxiliary_data_inverse to their inactive values, except for the current port which is not forced. This enables ICL Extraction to trace through OR-like combinational logic that combines several active-high auxiliary data pins into one single data signal. Please note that this logic will be represented by a OneHotDataGroup in the generated ICL. Therefore, the involved DataOutPorts must have the “Enable” property or an equivalent enabling condition to pass the ICL75 check.
------------------------	--

connection_rule_option (continued...)	<ul style="list-style-type: none">auxiliary_data_inverse — Identifies the DataOutPort as an active-low auxiliary data pin. When tracing from a port with connection rule option “auxiliary_data_inverse”, ICL Extraction asserts all ports on all instances declared as an auxiliary_data or auxiliary_data_inverse to their inactive values, except for the current port which is not forced. This enables ICL Extraction to trace through AND-like combinational logic that combines several active-low auxiliary data pins into one single data signal. Please note that this logic will be represented by a OneHotDataGroup in the generated ICL. Therefore, the involved DataOutPorts must have the “Enable” property or an equivalent enabling condition to pass the ICL75 check.auxiliary_data_enable — Identifies the DataOutPort as an auxiliary data enable pin. When tracing a port with a forced_low_output_port_list or a forced_high_output_port_list attribute pointing to a port declared as an auxiliary_data_enable or auxiliary_data_enable_inverse, ICL extract asserts all ports on all instances declared as an auxiliary_data_enable or auxiliary_data_enable_inverse to their inactive values, except for the ones referenced by the current port which are forced to their active values instead. This enables ICL extract to extract DataOut ports connected through an AND OR mux like this: (d1 & en1) (d2 & en2) (d3 & en3);auxiliary_data_enable_inverse — Identifies the DataOutPort as an auxiliary data enable inverse pin.must_connect_to_top_port — Allowed on input and output ports of an ICL instrument to indicate that this input or output port is only allowed to be connected to a primary input or output pin of the current design. For example, a ScanInPort is allowed to be sourced by a ScanOutPort of another instance. If this attribute is set to must_connect_to_top_port, it is only allowed to be sourced by a primary input port.
default_load_value	This attribute corresponds to the DefaultLoadValue that is specified on DataIn ports in the ICL. Possible values are: 0, 1, or X, where X is specified when the DataIn port does not have a specified default load value.
differential_inv_of	An icl_port attribute that specifies a functional clock. The value is computed from the optional DifferentialInvOf keyword of the ClockPort or ToClockPort statement.

direction	<p>Direction of the icl_port.</p> <p>Possible values are as follows:</p> <ul style="list-style-type: none"> • input • output
exclude_from_sdc	<p>An attribute that is used to exclude ICL feedthrough clocks that the tool identifies as reference clocks. When set to true, the extract_sdc command ignores the ICL port when tracing functional clocks and does not add a generated clock.</p> <p>The default is false.</p>
explicit_iwrite_only	<p>An attribute that is used to specify that the value of an ICL data object cannot be changed by the retargeter unless it is the target of an explicit iWrite command. In the ICL, the attribute can be specified inside DataInPort, DataOutPort, ScanRegister and DataRegister elements. The attribute value can be specified as a sized or unsized binary number using legal ICL numbers 0 and 1, where a 1 activates the described functionality on the corresponding port bit, while 0 represents ordinary behavior. If the attribute value is specified as a sized number, the size must match the size of the object. If the attribute value is specified as an unsized number, it is padded with 0s if necessary. The right-most bit (the least significant bit) of the attribute value represents the right-most bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on. You can also specify the attribute value using string values that are simply binary, decimal or hexadecimal numbers. Complex expressions are not allowed if the attribute is specified as a string, but concatenation is allowed. You can use this attribute to have full control over parts of the scan path configuration and prevent the retargeter from always using the shortest path.</p>
forced_high_dft_signal_list	<p>This attribute is specified inside the Port wrappers of the ICL modules and is used to list Dft signals that must be forced high during the tracing of the current port. The pins or ports that correspond to the listed Dft signals are simulated high and appended to the stable_after_setup simulation condition when tracing the associated port.</p>
forced_low_dft_signal_list	<p>This attribute is specified inside the Port wrappers of the ICL modules and is used to list Dft signals that must be forced low during the tracing of the current port. The pins or ports that correspond to the listed Dft signals are simulated low and appended to the stable_after_setup simulation condition when tracing the associated port.</p>

forced_low_output_port_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force low during the tracing of the current output port. The listed ports are simulated low and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.
forced_high_output_port_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force high during the tracing of the current output port. The listed ports are simulated high and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.
freq_multiplier	A positive integer value attribute at the icl_port. The value is computed from the optional FreqMultiplier keyword of the ICL ToClockPort statement. The default value is 0.
freq_divider	A positive integer value attribute at the icl_port. The value is computed from the optional FreqDivider keyword of the ICL ToClockPort statement. The default value is 0.
function_modifier	A string with value CaptureShiftClock or CaptureShiftClockInv that can be specified on a ToCaptureEnPort and a CaptureEnPort in ICL. When specified to CaptureShiftClock or CaptureShiftClockInv on a CaptureEnPort, the port can only trace to ToCaptureEnablePort or a CaptureEnPort with function_modifier also set to CaptureShiftClock or CaptureShiftClockInv. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClock, the port is driven by a gated version of TCK that is only active during the capture and shift cycles. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClockInv, the port is driven by a gated version of inverted TCK that is only active during the capture and shift cycles. The function_modifier attribute value sync_tester_clock can be applied to Clock ports and ToClock ports to indicate that the clock will be in sync with the tester clock.

icl_extraction_port_trigger_list	String that contains a list of ports that must be present in the same ICL module. An ICL port with this attribute does not serve as a starting point for ICL Extraction tracing from the very beginning. This port is used as starting point only if one of the ports listed in the attribute value has been reached by tracing from another instance or top module port.
ijtag_function	String that specifies the function of the icl_port; for example, "scan_in". This is the ICL port function string converted to lowercase and the words connected with underscores.
index	Integer that specifies the index of the icl_port that is already part of the name when the icl_port is part of a bus.
is_bus	Boolean that specifies whether the icl_port is part of a bus. This attribute is set to true when the icl_port is part of a bus.
is_valid	Boolean that specifies whether the ICL port still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
launch_edge	String attribute that is defined on a port with function ScanOut to reflect the LaunchEdge of the ScanOutPort in ICL.
master_module_name	String that specifies the master_name of the icl_module object the icl_port belongs to.
module_name	String that specifies the name of the icl_module object the icl_port belongs to.
name	String that specifies the name of the icl_port including its bus index when it is an element of a bus.
object_type	Returns the object data type "icl_port".
period	A string provides access to the "Period" specification of the ToClockPort in the ICL file.

read_latency	<p>An attribute that can be specified inside a DataRegister element. If present, the IJTAG retargeting engine will insert the specified latency as a delay before the corresponding DataRegister is read through an iRead operation. The delay is inserted right before the operation that captures value of the DataRegister. When an operation captures the value of multiple DataRegisters with the read_latency attribute, the delays are merged into a single delay. Use one of the following values for the read_latency attribute:</p> <ul style="list-style-type: none"> • tck <delay> : Indicates that the delay should last for <delay> cycles of the tester clock (TCK). • sck <icl_clock_port> <delay>: Indicates that the delay should last for <delay> cycles of the clock <icl_clock_port>. The clock has to be defined in the same module as the DataRegister with the read_latency attribute. The clock has to be previously traced with the iClock command. • time <time value>: Indicates that the delay should last for <time value>. The <time value> consists of a value and a time unit (s, ms, us, ns, fs or as), for example: 5us. When TCK is OFF, <delay> is converted to <delay>/tester_period TCK cycles rounded up to the next integer. If TCK is asynchronous free-running, TCK is ON and the <delay>/tester_period must be a multiple of the TCK ratio.
scan_interface_list	Lists all ScanInterface names that the port belongs to.
tessent_bisr_function	String that reflects the function of the ICL port for Tessent MemoryBIST's BISR ICL module. Legal values for this attribute are: Reset, MemoryDisable, Select.
tessent_clock_domain_labels	String that reports the label of clocks that were added prior to ICL extraction. The value is a list of label design pin/port. The attribute is added to ICL ports that represent asynchronous clocks, reference clocks, and branch clocks.
tessent_clock_periods	A string attribute that is set on ports of the current ICL design during ICL extraction. It reports the period that was set on the clock port with the “add_clocks -period” command prior to running ICL extraction. The value of this attribute is the “all” keyword followed by the period. The “all” keyword is used to reflect the fact that the period is used for all test modes. In the future, we may allow different periods to be specified for different test modes on the same pin, and this attribute will be able to reflect different periods for different modes by using the test mode name as the keyword.

tessent_common_tdr_source	An attribute that can be defined in any DataInPort wrapper inside the ICL Module wrapper to specify an arbitrary common source name for the DataInPorts. During the execution of the <code>create_dft_specification</code> command, one Tdr will be created with one DataOutPort for each unique source name such that all ports with the same common source name can be controlled by a single IJTAG signal. If "tessent_enable_group" is also specified, a separate Tdr will be created for each unique group.
tessent_default_procedure_name	An attribute that you can define inside a <code>ToClockPort</code> element of an ICL module to describe the name of an iProc that should be called by default to initialize the PLL or clock divider modeled by the ICL module. As you can see in Example 2 of the <code>ProcedureStep</code> section, the <code>create_icl_verification_patterns</code> command automatically detects that the memory BIST controller makes use of a clock that goes through the <code>ToClockPort</code> and automatically infers a <code>ProcedureStep</code> wrapper to initialize the PLL or clock divider associated to the <code>ToClockPort</code> . The <code>ProcedureStep</code> will include the <code>iProcArguments</code> wrapper if you also defined the <code>tessent_default_procedure_parameters</code> attribute.
tessent_default_procedure_parameters	An attribute that can be defined inside a <code>ToClockPort</code> element of an ICL module to describe the arguments that should be passed to the proc referenced by the <code>tessent_default_procedure_name</code> attribute
tessent_dft_function	A string attribute used to identify ICL ports having special meaning during DFT.
tessent_dft_signal_default_value_in_all_test	String attribute that defines the value the DFT signal is to have in all test modes. Its value is "1", or "0". See the description of the <code>register_static_dft_signal_names</code> command for more information. This attribute is mandatory when the <code>tessent_dft_signal_usage</code> attribute is set to <code>global_dft_control</code> and illegal otherwise.
tessent_dft_signal_name	A string attribute that defines the name of the DFT signal the ICL port corresponds to. This attribute is set on ports of the TDR created by <code>process_dft_specification</code> when the <code>add_dft_signals dft_signal_name -create_with_tdr</code> command is used. It is set on port of the current design when the <code>add_dft_signals dft_signal_name -source_node port_name -make_icl_port</code> command is used. If you want to have the output ports of your own TDR be automatically identified as the source of DFT signals, you must identify them with this attribute and three more attributes described below. See Example 2 in the <code>add_dft_signals</code> command for a complete example.

tessent_dft_signal_reset_value	A string attribute that defines the value the DFT signal is to have in functional mode. Its value is “1”, or “0”. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_name attribute is set on an ICL port.
tessent_dft_signal_scan_mode_type	A string attribute that defines the scan mode type of the DFT signal when the tessent_dft_signal_usage is set to scan_mode. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_usage attribute is set to scan_mode and illegal otherwise.
tessent_dft_signal_usage	A string attribute that defines the usage of the DFT signal. Its value is global_dft_control, logic_test_control and scan_mode. See the description of the register_static_dft_signal_names command for the meaning of each usage. This attribute is mandatory when the tessent_dft_signal_name attribute is set on an ICL port.
tessent_dft_signal_value_in_pre_scan_drc	A string attribute that defines the value the DFT signal is to be forced to during pre-DFT/pre-Scan DRC. Its value is “1”, “0”, or “x”. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_usage attribute is set to global_dft_control or logic_test_control.
tessent_enable_group	An attribute that can be defined in any DataInPort wrapper inside the ICL Module wrapper to specify arbitrary group names for the DataInPorts. During the execution of the create_dft_specification command, one Tdr will be created for each group of DataInPorts such that the group of ports can be controlled by IJTAG while leaving the ports in the other groups under functional control.

tessent_ignore_during_e14_drc	<p>This attribute controls the handling of ICL ports during the E14 check.</p> <p>If the test_setup procedure contains iCall commands, E14 also verifies the effect of the processed iRead and iWrite commands on the design. This verification can be suppressed by means of the “tessent_ignore_during_e14_drc” attribute.</p> <p>The attribute value can be specified as a sized or unsized binary number using legal ICL digits 0, 1 and X. If the attribute value is specified as a sized number, the size must match the size of the object. If the attribute value is specified as an unsized number, it is padded with 0s or Xs if necessary, dependent on the leftmost specified bit. The right-most bit (the least significant bit) of the attribute value refers to the right-most bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on. The value ‘1’ instructs Tessent Shell to ignore the bit of the port during the E14 check, the values ‘0’ and ‘X’ represent the default behavior.</p>
tessent_ignore_during_icl_verification	A string attribute used to identify an ICL module that is to be automatically excluded from the ICLNetworkVerify patterns or from the patterns created using the create_icl_verification_patterns command.
tessent_memory_bist_function	String that reflects the function of the ICL port for Tessent MemoryBIST’s ICL module of the memory module.
tessent_no_input_constraints	An attribute that affects only ICL ports of type DataInPort. If this attribute is not set on an ICL port with port function DataInPort or if it is set to “off”, then the port is automatically constrained to C0 or C1 at the end of the test_setup procedure, according to the last value that has been assigned to it during the IJTAG retargeting of the iCall and iMerge commands in the test_setup procedure. If the attribute is set to “on”, this behavior is suppressed.

tessent_use_in_dft_specification	<p>String attribute that you can set to off on an ICL Port to force the create_dft_specification command to ignore this DataIn or DataOut port. The attribute can be added in the ICL source description or set using the set_attribute_value command. The legal values for this attribute are: auto, false, auto_no_pi.</p> <p>When set or left to default to auto, the DataInPort will be driven by a TDR DataOutPort if it is not seen to be already sourced by a DataOutPort or a primary input of the current design. The DataOutPorts will be observed by a TDR DataInPort if it is not seen to be already observed by a DataInPort or a Primary output of the current design. See the description of the tessent_enable_group and tessent_common_tdr_source attributes above for a method to control the TDR assignment. The complete algorithm on how this works is described in the create_dft_specification command description. To have create_dft_specification ignore a complete ICL instance, use the “set_ijtag_instance_options -use_in_dft_specification off” command.</p> <p>When set to “auto_no_pi”, the DataInPort will be equipped with a local IJTAG control even if it is seen to already be sourced by a primary input. You want to use the “auto_no_pi” value for ports like the reset port of a PLL because they are often sourced by a system level reset port. Even though it is possible to toggle the system reset to reset the PLL, the effect of the reset pulse will be felt by the entire system and you may affect already running parts of the circuit. When you provide local controllability, it becomes possible to reset the PLL without affecting the rest of the chip. See the description of create_dft_specification for more information about how DataInPorts are handled during the creation of the DFT specification.</p>
write_latency	<p>An attribute that can be specified inside a DataRegister element. If present, the IJTAG retargeting engine will insert the specified latency as a delay after each iApply that contained an iWrite operation to the DataRegister. When the iApply contains write operations to multiple DataRegisters with the write_latency attribute, the delays are merged into a single delay.</p> <p>For details on the possible values of the write_latency attribute, see the documentation of the values for the read_latency attribute.</p>

Inheritance

Built-in attribute inheritance from associated icl_module objects:

- None

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Related Commands

get_attribute_list	get_iclock_option
get_attribute_value_list	get_name_list
get_icl_fanins	report_attributes
get_icl_fanouts	report_iclock
get_icl_ports	report_ijtag_logical_connections
get_iclock_list	

Port Specification

An ICL port object is an input, output, or inout interface of an ICL module, which is specified by a string. Examples of ICL port names are: “In1”, “ClkA”, and “Out3”.

Related Topics

[get_attribute_list](#)
[get_attribute_value_list](#)
[get_icl_fanins](#)
[get_icl_fanouts](#)
[get_icl_ports](#)
[get_iclock_list](#)
[get_iclock_option](#)
[get_name_list](#)
[report_attributes](#)
[report_iclock](#)
[report_ijtag_logical_connections](#)

icl_instance

An icl_instance is a single instantiation of an icl_module. No icl_instance object exists until the set_current_design command is executed.

The icl_instance object type interacts with attributes as follows:

- The icl_instance attributes are automatically visible as read-only attributes on corresponding icl_instances if the icl_instance object type does not have the same attribute registered.
- Attributes defined with the applies_to_child_instances option set to On automatically become visible to all child instances of the icl_instance object.
- The object type icl_instance treats different physical instances as different objects, even if they are associated with the same “[Instance](#)” construct in the ICL file (this can happen if the instances have the same parent module, but the parent module is instantiated several times). This introduces ambiguity during [write_icl](#), if an attribute is set to different values on different icl_instance objects associated with the same “Instance” construct. In case of ambiguity, the most recent usage of the commands [set_attribute_value](#) or [reset_attribute_value](#) determines the value of the attribute in the written ICL file.

For more information on attributes, see “[Attributes](#)” on page 3159.

Built-In Attributes

The icl_instance object type has the following attributes:

is_valid	Boolean that specifies whether the ICL instance still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the icl_instance (value of the name attribute) without the hierarchical path.
master_module_name	String that specifies the value of the master_name on the icl_module object associated with the icl_instance.
module_name	String that specifies the name of the ICL module associated to the icl_instance with the @PV# suffix when the instantiation uses parameter overrides.
name	String that specifies the full hierarchical instance path of the icl_instance with '.' as a hierarchy separator.
object_type	Returns the object data type “icl_instance”.
parent_instance	String that specifies the instance name of the ICL instance in which the pin is instantiated. It is null for pins on icl_instances found within the current design.

tessent_clock_domain_labels	String that reports the label of clocks that were added prior to ICL extraction. The value is a list of <i>label design pin/port</i> . The attribute is added to ICL instances that represent asynchronous clocks, reference clocks, and branch clocks.
tessent_design_instance	An attribute populated by ICL extract when extracting the top level ICL module. The attribute reflects the hierarchical instance path of the design instance corresponding to the ICL instance at the time of ICL extraction. The design instance path is relative to the current design at the time the ICL module was extracted which is always the parent ICL module of the ICL Instance. To get the complete design instance path of an ICL instance, use the <code>hierarchical_design_instance</code> attribute. ICL Extraction sets the <code>tessent_design_instance</code> attribute to the pre-synthesis instance names whenever those names are available.
hierarchical_design_instance	A string attribute that returns the complete hierarchical instance path of a design instance to the top. Unlike <code>tessent_design_instance</code> , <code>hierarchical_design_instance</code> is a read-only attribute that is computed internally and contains the complete instance path of a design instance. The path is computed only if the <code>tessent_design_instance</code> attribute is set for each instance in the hierarchy, otherwise an empty string is returned. The <code>tessent_design_instance</code> attribute, on the other hand, is an ICL attribute that is either user-specified or inserted by ICL extraction, and contains only the relative portion of the instance path between a design instance and the design module which matches the ICL module in which the ICL instance is instantiated.
tessent_ignore_during_icl_verification	A string attribute used to identify an ICL instance that is to be automatically excluded from the ICLNetworkVerify patterns or from the patterns created using the create_icl_verification_patterns command.

Inheritance

Built-in attribute inheritance from associated `icl_module` objects:

- `c0_constrained_internal_pin_list`
- `c0_constrained_pin_list`
- `c1_constrained_internal_pin_list`
- `c1_constrained_pin_list`
- `icl_extraction_time`
- `is_created`
- `partially_constrained_bus_pin_list`

- port_group_list
- tesson_instrument_type
- test_setup_procfile

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Related Commands

get_attribute_list	get_icl_scope
get_attribute_value_list	get_name_list
get_icl_instances	report_attributes

ICL Instance Specification

An ICL instance object is specified by a complete hierarchical ICL instance pathname. For example: “/u1/u2/u3”.

Related Topics

- [get_attribute_list](#)
[get_attribute_value_list](#)
[get_icl_instances](#)
[get_icl_scope](#)
[get_name_list](#)
[report_attributes](#)

icl_pin

An `icl_pin` is an object on an `icl_instance` that can be connected to other `icl_pin` or `icl_port` objects. No `icl_pin` object exists until the `set_current_design` command is executed.

The `icl_pin` object type interacts with attributes as follows:

- Attributes defined on `icl_port` objects are automatically visible as read-only attributes on the associated `icl_pin` objects unless the attribute is also registered for the `icl_pin` object type. This is the case for the `name` attribute which is defined for both `icl_port` and `icl_pin` object types.

For more information on attributes, see “[Attributes](#)” on page 3159.

Built-In Attributes

The icl_pin object type has the following attributes

base_name	String that specifies the name of the pin without the leaf name.
freq_multiplier	A positive integer value attribute at the icl_pin. The value is computed from the optional FreqMultiplier keyword of the ICL ToClockPort statement. The default value is 0.
freq_divider	A positive integer value attribute at the icl_pin. The value is computed from the optional FreqDivider keyword of the ICL ToClockPort statement. The default value is 0.
is_valid	Boolean that specifies whether the icl_pin still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the ICL pin (value of the name attribute) without the hierarchical path.
name	String that specifies the full hierarchical name of the icl_pin with '.' as a hierarchy separator.
object_type	Returns the object data type "icl_pin".
parent_instance	String that specifies the name of the parent icl_instance of the icl_instance on which the icl_pin is on.
tessent_bisr_power_domain_name	This attribute is only specified for ICL pins of the fuse box controller, which is part of the memory repair technology of Tessent MemoryBIST. The attribute is used to identify the power domain group label for each pin.
tessent_ignore_during_icl_verification	A string attribute used to identify an ICL module that is to be automatically excluded from the ICLNetworkVerify patterns or from the patterns created using the create_icl_verification_patterns command.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, or "" (empty string). The default is the empty string which means the port is not tied to a value.

Inheritance

Built-in attribute inheritance from associated icl_port objects:

- bus_left_index
- bus_name
- bus_range_direction
- bus_right_index

- bus_width
- direction
- ijtag_function
- index
- is_bus
- master_module_name
- module_name
- period
- port_group_list

User-defined attributes are inherited. See “[Attribute Inheritance Behavior](#)” on page 3170.

Pin Specification

An ICL pin object is specified by a complete hierarchical ICL pin pathname including the ICL pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Related Topics

- [get_attribute_list](#)
- [get_attribute_value_list](#)
- [get_icl_fanins](#)
- [get_icl_fanouts](#)
- [get_icl_pins](#)
- [get_icl_instances](#)
- [get_iclock_list](#)
- [get_iclock_option](#)
- [get_name_list](#)
- [report_attributes](#)
- [report_ijtag_logical_connections](#)

Scan Data Model

This section describes the scan models used for Hierarchical Scan Insertion and their built-in attributes.

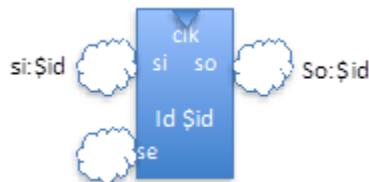
Scan Element Object Type	3144
Scan Chain Family Object Type.....	3150
Scan Mode Object Type	3151

Scan Element Object Type

The Scan Element object type is used to represent all scan path objects making up the scan segments and the scan chains.

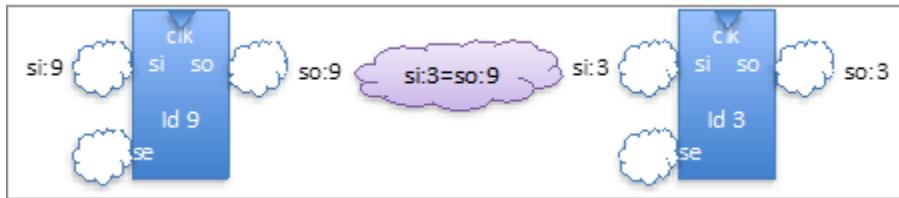
A scan element is either hierarchical (for example, scan chain, scan segment, shift register) or a leaf node, which models a single scan path of a cell or blackbox. The most common leaf scan element model is a single-bit scan cell, but multi-bit leaves like a flop-tray or a multi-chain cell can be modeled with one or more scan elements respectively. After DRC, there is one “active” scan element per scannable path of each cell, per declared scan segment and per scan chain. Most scan element leaves are populated from DRC results, but some get inferred by the planned insertion of certain types of modules like dedicated wrapper cells; these scan elements are deemed “virtual” as they do not exist yet but are assumed to exist for analysis and planning purposes.

The figure below is a simplified graphical representation of a scan_element. \$id in si:\$id and so:\$id means that the group is taken from the id (node group) of the associated element. Other signals are classified into three different groups: at_speed_control, static_control, and clock. The scan element object holds a unique value for each of those group IDs.



The si:\$id of one scan element connects to the si signal of its parent scan element when it is the first child element. It connects to the so of the previous element when it is not the first element. The so:\$id of a hierarchical scan element connects to the so of its last child. The next figure

illustrates how the scan path equation “si:3 = so:9” is used to create a scan path between scan elements 9 and 3. Similarly, equations are used to control the scan enable of each scan element.



If specified, the si equation must include the signal type "si" as one of its inputs; similarly, the so equation must include the signal type "so" as one of its input. If specified, the clk equation must include the signal type "clk" or the operator <src> as one of its inputs.

Table 8-2. Built in attributes on Scan Element (SE) Object Type

Attribute name	Default	Description
class		standard, core, wrapper
clock_domains		The clock domain name(s) of this element.
cluster_name		The value of the cluster_name attribute, if any.
exists		Indicates if the scan element exists in the design netlist; a false value signifies that the element has not been implemented yet
has_children		Indicates if a hierarchical scan element has sub-scan elements defining it.
id		This is a unique integer that uniquely identifies a “scan_element” object. This id is only persistent within one run while in the analysis or insertion state.
is_non_scannable	False	Boolean that specifies that the scan element is non-scannable. Default is false.

Table 8-2. Built in attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
is_non_scannable_reason		<p>A string that specifies why the scan_element is non-scannable, as follows:</p> <ul style="list-style-type: none">• user_specified — The object was specified to be non-scannable using the add_nonscan_instances command.• inferred_from_drc — The object was inferred to be non-scannable during DRC either because it had an S-rule violation or it is part of a hard_module yet excluded from the traced scan chains.• imported_from_tcd_scan — The object was inferred to be nonscannable because it has a Core/Scan TCD description and the property drc_from_boundary_only is equal to 1.• imported_from_icl — The instance was inferred to be nonscannable because it has a matched ICL description and the attribute keep_active_during_scan_test is not specified as false.• imported_from_bscan — The instance was inferred to be nonscannable because it is part of the boundary scan chain.

Table 8-2. Built in attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
is_non_scannable_reason (continued)		<ul style="list-style-type: none"> • missing_scan_model <ul style="list-style-type: none"> — The instance was inferred to be nonscannable because it does not have a scan flop equivalent mapping. This only happens in dft-scan context. • user_specified_from_net <ul style="list-style-type: none"> — The instance is non scannable because the add_nonscan_instances -rtl_reg command was used on a net which happens to be an RTL register. The instance happens to be the flip-flop instance that the tool created during quick synthesis for that RTL register. • If is_non_scannable is false, then is_nonscannable_reason returns an empty string. The scan_element also inherits is_non_scannable_reason values from its associated instance (see instance attributes).
length		This is the length of the scan element object (number of master sequential cells recursively found down the hierarchy).
name		This is an optional name that was given when defining the “scan_element” object. This name is not necessarily unique and may be null.
no_si_pipeline		Boolean indicating that no pipelining element would get inserted at the beginning of a new scan chain if this element is the first element in the chain.

Table 8-2. Built in attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
no_si_pipelining_reason		A string that specifies why no_si_pipelining is set for a scan_element, as follows: <ul style="list-style-type: none">• imported_from_tcd_scan<ul style="list-style-type: none">— A .tcd_scan file specified no_si_pipelining on this scan_element.• user_specified<ul style="list-style-type: none">— The object specified no_si_pipelining by setting the specified_no_si_pipelining attribute.• If no_si_pipelining is false, then is_nonscannable_reason returns an empty string.
power_domain_island		Returns the name of the power domain island associated to the scan_element. This attribute is inherited for its parent design object if a UPF or CPF file has been read in. For a definition of Power Domain Island , refer to the <i>Tessent Glossary</i> .
power_domain_name		The value of the power_domain_name attribute, if any. For a definition of Power Domain , refer to the <i>Tessent Glossary</i> .
si_clock_domain		Clock domain for the first scan cell.
si_clock_edge		Capture edge for the first scan cell.
so_clock_domain		Clock domain for the last scan cell.
so_clock_edge		Capture edge for the last scan cell.
specified_no_si_pipelining		An attribute that prevents a pipelining element to get inserted at the beginning of a new scan chain if this element is the first element in the chain, even if the scan insertion options specifies -si_pipelining for that chain.

Table 8-2. Built in attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
specified_wrapper_type	Off	<p>An attribute that forces the wrapper type of a scan_element. Available values are: off, input, output, and generic. The default is off.</p> <ul style="list-style-type: none"> The result of the analyze_wrapper_cells command, which combines the specified_wrapper_type values with the inferred values, are reflected in the wrapper_type and wrapper_type_reason attributes described below. When set or left to default to off, the wrapper_type value of each scan_element object is computed by the analyze_wrapper_cells command. When you set the specified_wrapper_type attribute to input, output, or generic, the specification is kept during the analysis. If you used the input value, the analyze_wrapper_cells command will automatically find all flip-flops fanning out to the scan_element you forced to input type and infer them as output wrapper type.
state		ignored unusable unusable_child_mode unresolved_child_mode usable usable_indirectly. Only “usable” scan elements can be specified as part of a population when defining a chain/segment using the “-include_...” option.
type		chain segment leaf_cell shift_register transparent_leaf_cell inferred
used		Indicates if the scan element is used in at least one mode

Table 8-2. Built in attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
wrapper_type		input output generic ""
wrapper_type_reason		An attribute that describes why the wrapper has a given type. <ul style="list-style-type: none"> • inferred_from_analysis <ul style="list-style-type: none"> — Indicates the type was inferred by the analyze_wrapper_cells command. • inferred_from_clock_source <ul style="list-style-type: none"> — Means the instance is an OCC, the fast_clock port of the OCC is not directly sourced by a PI (that is, a PLL or clock divider), and the OCC drives at least one flop of type input or output wrapper. The wrapper_type is always generic when wrapper_type_reason is inferred_from_clock_source. • user_specified <ul style="list-style-type: none"> — Indicates the specified_wrapper_type attribute was explicitly set on the flop or clock gater instances.

Scan Chain Family Object Type

The scan_chain_family is an intelligent container that controls the allocation of new scan chains from a specific population of scan element.

The command create_scan_chain_family is used to generate new chain families; the related commands delete_scan_chain_families and get_scan_chain_families are also available. The current introspection is currently very limited, but will eventually include the various scan insertion options specified for this particular population sample.

Table 8-3. Built in Attributes on Scan Chain Family Object Type

Attribute name	Default	Description
name		This is an optional name than was given when defining the “scan_chain_family” object. This name is not necessarily unique and may be null.

Table 8-3. Built in Attributes on Scan Chain Family Object Type (cont.)

Attribute name	Default	Description
id		This is a unique integer that uniquely identifies a “scan_chain_family” object. This id is only persistent within one run while in the analysis or insertion state.

Scan Mode Object Type

The scan_mode object controls the allocation of scan chains for a specific scan mode. The scan_mode object controls the allocation of scan chains for a specific scan mode.

The hierarchical scan engine allows you to specify any number of scan modes. A mode typically applies to an entire scan element population or a large subset (for example, wrapper elements), and is typically populated by including scan_element objects directly and/or scan_chain_family objects.

Table 8-4. Built-in Attributes on Scan Mode Object Type

Attribute Name	Default	Description
name		This is an optional name than was given when defining the “scan_mode” object. This name is not necessarily unique and may be null.
id		This is a unique integer that uniquely identifies a “scan_mode” object. This id is only persistent within one run while in the analysis or insertion state.

Test Point Data Model

This section describes the Test Point data model and its built-in attributes.

Test points are inserted into the design to increase controllability and/or observability of otherwise hard to reach locations. For more information, see “[What are Test Points?](#)” in the *Tessent Scan and ATPG User’s Manual*.

Table 8-5. Built in Attributes on Test Point Object Type

Attribute name	Default	Description
name		A unique name that was given when defining the “test point” object.
id		A unique integer that uniquely identifies a “test_point” object. This id is only persistent within one run while in the analysis or insertion state.
location		A string representing the full design path to the observed/controlled gate.
shared_id		An integer indicating the test points that are driven by the same flop. This integer is unique per sharing group.
type		A string that specifies the type of test point. The possible values are: <ul style="list-style-type: none"> • control point • observe point
control_point_gate_type		A string that specifies the type of control point. The possible values are: <ul style="list-style-type: none"> • AND Control Point • OR Control Point
origin		A string that specifies how the test point was created. The possible values are: <ul style="list-style-type: none"> • user_specified — created by the user or imported from an external tool/script • design_analysis — created by the tool with the analyze_test_points command
power_domain_name		A string representing the power domain name of the test point location.
power_domain_island		A string representing the power domain island in which the test point is placed.
clock_connection_name		A string representing the full design path to the clock that is driving the test point flop.
enable_connection_name		A string representing the full design path to the connection that enables the test point.
flop_location		A string representing the full design path to the observed/controlled flop instance.

Configuration Data Model

This section describes the Configuration data model and its built-in attributes.

config_csv_wrapper	3153
config_data_wrapper	3154
config_property	3155
config_repeatable_property	3155
config_wrapper	3156

config_csv_wrapper

This section describes the built-in attributes for the config_csv_wrapper object type.

You use this metadata syntax to define a config_csv_wrapper in the [CsvWrapper](#) that contains comma separated value lists.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it will reflect that the node is open with a true value. If you save the configuration data to file using write_config_data , you will see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data will restore the value of this attribute where the wrapper starting with +{ will have this attribute true and those open with { will have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.

name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_csv_wrapper' for this object type.

config_data_wrapper

This section describes the built-in attributes for the config_data_wrapper object.

You use this metadata syntax to define a config_data_wrapper in the [DataWrapper](#) that contains comma separated value lists.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it will reflect that the node is open with a true value. If you save the configuration data to file using write_config_data , you will see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data will restore the value of this attribute where the wrapper starting with +{ will have this attribute true and those open with { will have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.

object_type	Returns the object type of the configuration element, namely 'config_data_wrapper' for this object type.
-------------	--

config_property

This section describes the built-in attributes of the config_property object.

You use this metadata syntax to define a config_property wrapper in the [Property](#) wrapper that contains comma separated value lists.

Built-In Attributes

file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_property' for this object type.
value	String that returns the value of the property the same way get_config_value does. The value attribute is useful when you want to filter a collection of config property objects based on their current values.

config_repeatable_property

The section describes the built-in attributes for the config_repeatable_property object.

You use this metadata syntax to define a config_repeatable_property wrapper in the [RepeatableProperty](#) wrapper that contains comma separated value lists.

Built-In Attributes

file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_repeatable_property' for this object type.
value	String that returns the value of the property the same way get_config_value does. The value attribute is useful when you want to filter a collection of config property objects based on their current values.

config_wrapper

This section describes the built-in attributes for the config_wrapper object.

You use this metadata syntax to define a config_wrapper in the [Wrapper](#) that contains comma separated value lists.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it will reflect that the node is open with a true value. If you save the configuration data to file using write_config_data , you will see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data will restore the value of this attribute where the wrapper starting with +{ will have this attribute true and those open with { will have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element , the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_wrapper' for this object type.
tree_node_state	String that specifies the tree node state. Possible values are as follows: <ul style="list-style-type: none">• default• pass• maybe• fail

Chapter 9

Attributes

Attributes are properties assigned to any object type. Attributes have names and values to associate relevant data with those objects. Attribute names may only include alphanumeric and underscore characters.

This section provides information on the following topics:

For detailed information on the attributes associated with each object type, see “[Data Models](#)” on page 3055.

Attribute-Related Commands	3159
Attribute Types	3161
Pre-Defined Attributes	3161
User-Defined Attributes	3161
Attribute Value Types	3161
Attribute Filtering Equation Syntax	3162
Glob and Regular Expression Pattern Matching Syntax	3164
Hierarchical Name Matching With Escaped Identifiers	3167
Attribute Inheritance Behavior	3170
Inheritance Rules	3170
DFT Test Logic Related Attributes (no_control_point and no_observe_point)	3171

Attribute-Related Commands

You use the attribute-related Tesson Shell commands to create, manipulate, and query objects' attributes.

Table 9-1. Commands for Manipulating Attributes

Command	Description
get_attribute_list	Returns an alphabetically sorted Tcl list of attribute names.
get_attribute_option	Retrieves the current setting of an attribute's option.
get_attribute_value_list	Retrieves the value of an attribute on the specified design objects.
get_name_list	Returns the name(s) of the specified object(s).
register_attribute	Registers a new user-defined attribute.

Table 9-1. Commands for Manipulating Attributes (cont.)

Command	Description
report_attributes	Creates a human readable report for the set of attributes based on the usage options.
reset_attribute_value	Resets an attribute to its default value for the design objects specified in obj_spec.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in object_spec.
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.

Attribute Types

Tessent Shell supports two types of attributes: pre-defined attributes and user-defined attributes. Different object types have different pre-defined attributes. You can also create and specify user-defined attributes.

Pre-Defined Attributes	3161
User-Defined Attributes	3161

Pre-Defined Attributes

Pre-defined attributes exist on the object itself in the internal database. Because of this, pre-defined attributes do not need to be registered. Different object types have different pre-defined attributes. By default, these attributes are read-only.

Each data model has a list of pre-defined attributes that are described in sections “[Hierarchical Design Data Model](#)”, “[Flat Design Data Model](#)”, and “[ICL Data Model](#)”.

User-Defined Attributes

User-defined attributes are registered with the `register_attribute` command before they can be used. The registration operation defines the attribute’s value type and its default value and determines which design object type the attribute belongs to.

Attribute Value Types

Both pre-defined and user-defined attributes have an attribute value type.

Table 9-2. Attribute Value Types

Type	Description
Boolean	Case-insensitive. A Boolean. <ul style="list-style-type: none">Accepts any of the following for true: 1, true, yes, on.Accepts any of the following for false: 0, false, no, off
Integer	Case-insensitive. An integer.
Double	Case-insensitive. Double-precision floating-point number.
String	Case-insensitive. A sequence of alphanumeric characters.

Attribute Filtering Equation Syntax

An attribute filtering equation is a series of relations joined together with the `&&` (AND) and `||` (OR) operators. Parentheses are also supported. A basic relation compares or matches the value of an attribute with a constant expression through a relational operator.

Relational Operators

The relational operators are as follows:

<code>==</code>	Equal
<code>!=</code>	Not equal
<code>></code>	Greater than
<code><</code>	Less than
<code>>=</code>	Greater than or equal to
<code><=</code>	Less than or equal to
<code>=~</code>	Matches pattern
<code>!~</code>	Does not match pattern

Basic Relational Rules

The basic relational rules are as follows:

- Compare string attributes using any operator.
- Do not compare numeric attributes using pattern matching operators (`=~` or `!~`).
- Compare Boolean attributes using only `==` and `!=`. The only possible value is true or false.
- Using an attribute without a relational operator in an equation is equivalent to saying `att == true` for a Boolean attribute or `att != <defaultValue>` for other types. You can also invert this default behavior by putting a `!` in front of the attribute name. For example:

```
is_something && !is_something_else
```

translates to:

```
is_something != <default> && is_something_else == <default>
```

which mean the following when the attributes are of Boolean type:

```
is_something == true && is_something_else == false
```

Pattern Matching in the Filtering Attribute Equation

By default, the `=~` and `!~` filter operators use simple wildcard pattern matching (glob) with the `“*”` and `“?”` wildcards. When the `-regexp` option is specified for any command, the complete Posix Extended Regular Expression syntax is supported.

When you use the `-regexp` option, you should pay special attention to how you quote filter expressions. We recommend using rigid quoting with braces around regular expressions.

Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding `“.*”` to the beginning or end of the expression as necessary.

Refer to “[Glob and Regular Expression Pattern Matching Syntax](#)” for a complete description of the Tesseract Shell pattern matching mechanism and a detailed description of the regular expression syntax.

Filter Equation Examples

This example shows different filtering attribute equations used to filter instance objects:

```
register_attribute -name Att1 -value_type boolean -object_types instance
register_attribute -name Att2 -value_type integer -default -1
register_attribute -name Att3 -value_type string -default ""

set_attribute_value u1 -name Att1
set_attribute_value u1 -name Att2 -value 3
set_attribute_value u1 -name Att3 -value "aba"

set_attribute_value u2 -name Att1
set_attribute_value u2 -name Att3 -value "abb"

#Matches only u1 because Att2 has default value on u2 so Att2 returns
#false

get_instance -filter {Att1 && Att2}
{u1}

#Matches both because Att3 starts with a in both

get_instance -filter {Att1 && Att3 =~ "a*”}
{u1 u2}

#Matches u2 because Att2 defaults to -1.

get_instances -filter {Att2 < 2}
{u2}
```

Glob and Regular Expression Pattern Matching Syntax

The introspection commands perform pattern matching in two distinct ways with subtle usage differences. The first way is with the *name_patterns* argument and it is specified as a list of patterns to match against object names. The second one is the use of the `=~` and `~!` operators within a filter attribute equation.

When specifying *name_patterns*, the hierarchy separator (`/`) is used to automatically split the matching patterns so that each pattern can be matched against the leaf name of each level of hierarchy separately. For example, the command “`get_instance a*/b*`” means to match the leaf instance starting with `b` in each instance whose name starts with `a`. Refer to section “[Hierarchical Name Matching With Escaped Identifiers](#)” for ways to deal with that in the presence of escape identifiers, especially when the escaped identifiers contains slashes.

When specifying the filter attribute equation, the `/` is treated as a literal within the pattern and the entire value of the attribute is searched by the entire pattern. For example, the command “`get_instance -filter {name =~ "a*/b*"}`” would match instance `a1/b1` as well as `a1/c1/b1` because `a*` matches “`1/c1`” as the `/` is just another literal character to match.

In regular expressions syntax, most characters are treated as literals. That is, they match only themselves (for example, `a` matches “`a`”). The exceptions are called meta characters and are listed in [Table 9-3](#)

Table 9-3. Posix Extended Regular Expression Metacharacters

Meta Character	Description
.	The dot matches any single character. Within bracket expressions, the dot character matches a literal dot. For example: <code>a.c</code> matches “ <code>abc</code> ” and so on but <code>[a.c]</code> matches only “ <code>a</code> ”, “ <code>.</code> ”, or “ <code>c</code> ”.
[]	A bracket expression. Matches a single character that is contained within the brackets. For example, <code>[abc]</code> matches “ <code>a</code> ”, “ <code>b</code> ”, or “ <code>c</code> ”. <code>[a-z]</code> specifies a range which matches any lowercase letter from “ <code>a</code> ” to “ <code>z</code> ”. These forms can be mixed: <code>[abcx-z]</code> matches “ <code>a</code> ”, “ <code>b</code> ”, “ <code>c</code> ”, “ <code>x</code> ”, “ <code>y</code> ”, or “ <code>z</code> ”, as does <code>[a-cx-z]</code> . The <code>-</code> character is treated as a literal character if it is the last or the first character within the brackets: <code>[abc-]</code> , <code>[-abc]</code> . Note that backslash escapes are not allowed. The <code>]</code> character can be included in a bracket expression if they are the first character: <code>[]abc</code> . The <code>[</code> character can appear anywhere.
[^]	Matches a single character that is not contained within the brackets. For example, <code>[^abc]</code> matches any character other than “ <code>a</code> ”, “ <code>b</code> ”, or “ <code>c</code> ”. <code>[^a-z]</code> matches any single character that is not a lowercase letter from “ <code>a</code> ” to “ <code>z</code> ”. As above, literal characters and ranges can be mixed.

Table 9-3. Posix Extended Regular Expression Metacharacters (cont.)

Meta Character	Description
^	Matches the starting position of the string. In Tesson shell, all matching is rooted which means the ^ character is inferred at the beginning of every expression when it is not already specified. For example, abc.* is equivalent to ^abc.* and both mean to match any string starting with abc.
\$	Matches the ending position of the string. In Tesson shell, all matching are rooted which means that the \$ character is inferred at the end of every expression when it is not already specified. For example, .*abc is equivalent to .*abc\$ and both mean to match any string ending with abc.
()	Defines a marked expression. The string matched within the parentheses can be recalled later (see next entry, \n). A marked expression can also be used to specify a sequence of characters that must be matched some number of times by following it by a *, + or {m.n} sequence. For example, (ab)* matches "", "ab", "abab", "ababab", and so on.
\n	Matches what the nth marked expression matched, where n is a digit from 1 to 9. For example (ab).*\1 matches any string that starts and ends with either a or b, such as aa, aca, aba, or bab but not acb and bca. A second example (ab)(cd).*\2\1 matches acffca but not acffaa because the second marked expression matched c and the first matched a.
*	Matches the preceding element zero or more times. For example: ab*c matches "ac", "abc", "abbbc", and so on. [xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.
?	Matches the preceding element zero or one time. For example: ba? matches "b" or "ba".
+	Matches the preceding element one or more times. For example: ba+ matches "ba", "baa", "baaa", and so on.
	The choice operator matches either the expression before or the expression after the operator. For example: abc def matches "abc" or "def".
{m,n}	Matches the preceding element at least m and not more than n times. For example: a\{3,5\} matches only "aaa", "aaaa", and "aaaaa". (ab){2,3} matches "abab" and "ababab".

Table 9-4 the escapes character that have special meaning in Posix regular expression syntax:

Table 9-4. Special Characters in Regular Expressions

Escaped character	Description
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab

Table 9-5 lists the pre-defined special character sets that can be used in regular expressions:

Table 9-5. Special Characters Set in Regular Expression

Symbol	Content	Description
[:alnum:]	[A-Za-z0-9]	Alphanumeric characters
[:alpha:]	[A-Za-z]	Alphabetic characters
[:blank:]	[\t]	Space and tab
[:cntrl:]	[\x00-\x1F\x7F]	Control characters
[:digit:]	[0-9]	Digits
[:graph:]	[\x21-\x7E]	Visible characters
[:lower:]	[a-z]	Lowercase letters
[:print:]	[\x20-\x7E]	Visible characters and space character
[:punct:]	[!#\$%&'()*+,.:/;<=>?@^\`{ }~-]	Punctuation characters
[:space:]	[\t\r\n\f]	Whitespace characters
[:upper:]	[A-Z]	Uppercase letters
[:xdigit:]	[A-Fa-f0-9]	Hexadecimal digits

Example 1

The following list of examples illustrates the use of meta characters in matching expressions:

- .at matches any three-character string ending with “at”, including “hat”, “cat”, “bat”.
- [hc]at matches “hat” and “cat”.
- [^b]at matches all strings matched by .at except “bat”.
- ^[hc]at matches “hat” and “cat”, but only at the beginning of the string or line.
- [hc]at\$ matches “hat” and “cat”, but only at the end of the string or line.

- `\[\]` matches any single character surrounded by “[” and “]” since the brackets are escaped. For example: “[a]” and “[b]”.
- `[hc]+at` matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, and so on, but not “at”.
- `[hc]?at` matches “hat”, “cat”, and “at”.
- `[hc]*at` matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, “at”, and so on.
- `cat|dog` matches “cat” or “dog”.
- `[:digit:]]+ matches “1”, and “32” but not “a1”.`

Example 2

The following example matches all leaf instance names between 3 and 4 characters in length.

```
get_instances -filter { leaf_name =~ "[[:alnum:]]{3,4}" } -regexp
{u1/u2/abc u1/u3/ab_1}
```

Example 3

The following example matches all instances having a leaf instance name with a prefix matching the value within a Tcl variable followed by 1 or many digits. In the example, the attribute equation cannot be enclosed in braces because it would prevent the Tcl shell from replacing the Tcl variable. Instead, the attribute equation is enclosed in double quotes and all special characters within the expression are escaped.

```
set prefix "MyString_"
get_instances -filter "leaf_name =~ \"$${prefix}\[[:digit:]\]+\\" -regexp
{u1/u3/MyString_1 u1/u2/MyString_75}
```

Example 4

The following example looks for all leaf instances with a prefix equal to abc inside all instances starting with u or U followed by one or more digits. Notice that the / is automatically inferred as a hierarchy separator causing the expression to be split in two and applied to each level of hierarchy separately.

```
get_instance {[Uu][[:digit:]]+/abc.*} -regexp
{u3/abcd U43/abc}
```

Hierarchical Name Matching With Escaped Identifiers

The slashes inside a name_patterns argument have special meaning; they are interpreted as hierarchy separators unless they are within escaped identifiers. For example, consider an instance whose name has two escaped identifiers as part of its hierarchical name: “u1/u2/u3 /

u4/u5 /insta". In Tessent Shell, all attribute, introspection, and editing commands can reference such a name with and without the escape characters and trailing spaces.

For more information, refer to "[Glob and Regular Expression Pattern Matching Syntax](#)" on page 3164.

Note

 Legacy commands (commands that existed prior to v9.6) only support names from which the escape characters and trailing spaces have been removed. However, the legacy commands are upgraded to accept collections of objects instead of a simple list of names so you can pass in directly the output of commands such as get_pins and get_instances.

When the escape identifier and trailing spaces are removed, the example path "u1\|u2/u3 \|u4/u5 /insta" becomes "u1/u2/u3/u4/u5/insta". As you can see, the four levels of hierarchy appear as six levels because the leaf instance names \u2/u3 and \u4/u5 look like two levels each.

Matching "u1/u2/u3/u4/u5/insta" against "u1\|u2/u3 \|u4/u5 /insta" requires searching for u1\|u2/u3 if no matches are found for u1/u2. You must apply this guessing approach recursively which can be expensive when the escaped identifiers contain many slashes. More importantly, pattern matching on these names is difficult because the / is interpreted as a hierarchy separator which incorrectly splits the name_patterns into the wrong patterns to match against the level of hierarchies. For these reasons, we recommend you keep the escape identifiers in the name_patterns argument to prevent escaped slashes from being interpreted as hierarchy separators.

The following two examples both successfully match the instance with name "u1\|u2/u3 \|u4/u5 /insta":

```
get_instances {{u1\|u2/u3 \|u4/u5 /insta}}
get_instances {{u1\|u[0-9]+/u[0-9]+ \|u[0-9]+/u[0-9]+ /insta}} -regexp
```

Because the get_instances command accepts a list of pattern names as input, two sets of braces are used to enclose the name_patterns argument. The Tcl shell performs a first evaluation to get the list items and consumes one set of braces; it performs a second evaluation on each item and consumes the second set of braces. Without two sets of braces, you would need to escape both bracket characters "[" and "]" to keep them as part of the regular expression and prevent them from being interpreted as a Tcl command. Notice how the \ that is part of the escaped identifier is escaped by a second \ when the -regexp option is used. This is because the \ is a special character in Posix regular expression syntax and it must be escaped to be treated as a literal.

In the second example, "get_instances {{u1\|u[0-9]+/u[0-9]+ \|u[0-9]+/u[0-9]+ /insta}} -regexp", the *leaf instance* matching insta, inside leaf instances matching "\u[0-9]+/u[0-9]+", inside leaf instances matching "\u[0-9]+/u[0-9]+", inside leaf instance matching u1 is returned. Notice how the slashes within the escaped identifiers were not interpreted as hierarchy separators but as literals within the patterns.

The parser terminates the escape identifier string when it encounters a white space. For this reason, your patterns must not include any spaces; there is no reason to include spaces when matching instance path names.

If you need the Tcl shell to replace the Tcl variable within the regular expression, as in the example below, you must use double quotes instead of the inner set of braces and escape every special character. The `\` became `\\\` because the Tcl interpreter will replace `\\\` by `\`, and the `\` will be interpreted as a `\` literal by the regexp parser.

Having to escape all special characters is very error prone. We recommend that you use the double sets of braces as shown above unless your pattern contains Tcl variables that must be replaced by the shell as shown below.

```
set range "0-9"
get_instances {"u1\\\\\\u\\[$range]+/u\\[$range]+ /\\\\u\\[$range]+/u\\[$range]+ /insta"} \
-regexp
```

Attribute Inheritance Behavior

By default, attributes with the applies_to_child_instances option set to On are visible to all child instances below the current module. When you query the attribute value, the tool first looks to see if it is explicitly set for the specified instance. If the attribute is not set, it then traverses up the hierarchy and checks to see if the attribute is set in any of the parent instances. The first explicitly specified attribute value found will be returned to you. The default value will be returned if you did not explicitly set it at any level of the hierarchy.

Inheritance Rules [3170](#)

DFT Test Logic Related Attributes (no_control_point and no_observe_point) [3171](#)

Inheritance Rules

Attribute inheritance follows specific rules.

Normal Attribute Inheritance Rule

If the attribute is not registered for the current object type, the attribute can be inherited from a different object type. The individual data model descriptions (such as, Module, Instance, Port, and so on) provide an “Inheritance” section that lists the build-in inherited attributes.

Special Boolean Inheritance Rule

If a boolean attribute value is not specified for the current object, the attribute value can be inherited from an object of a different object type. To determine if a given attribute is specified for the current object, use the “[get_attribute_value_list -is_specified](#)” command and switch.

The following example demonstrates the boolean inheritance with a pin and a port, specifically the pin inheriting the attribute from its port:

```
register_attribute -name myboolattr -value_type boolean -object_types [list port pin]
set pin [get_pins i1/a]
set port [get_ports -of_pins $pin]
set_attribute_value $port -name myboolattr -value true // value is set to true on the port
set_attribute_value $pin -name myboolattr -value false // value is set to false on the pin
puts [get_attribute_value_list $pin -name myboolattr]
// no inheritance since the value is specified on the pin
false

report_attributes $pin
Attributes defined for object 'i1/a':
Name          Value  Inheritance
...
myboolattr    false  -
...
```

```

puts [get_attribute_value_list $pin -name myboolattr -is_specified]
1

reset_attribute_value $pin -name myboolattr // unspecified the value on the pin
puts [get_attribute_value_list $pin -name myboolattr] // value is inherited from the port
true

puts [get_attribute_value_list $pin -name myboolattr -is_specified]
0

report_attributes $pin

Attributes defined for object 'i1/a':
Name          Value   Inheritance
-----
...
myboolattr    true    port
...

```

Attribute Inheritance Reporting

Using the `report_attributes` command with a collection of objects lists an Inheritance column that shows if the value is set on the current object, or if the value is inherited from another object type. For example:

```

report_attributes [get_scan_elements -class wrapper]

Attribute Definition Report
Attributes defined for object '/\cnt_s_reg[0] ':
Name          Value   Inheritance
-----
at_speed_control_group 2
cell_type      scan_cell

```

Attributes defined with the `applies_to_child_instances` option set to “on” show the value that is inherited from a parent instance or `icl_instance`.

DFT Test Logic Related Attributes (no_control_point and no_observe_point)

The `no_control_point` and `no_observe_point` attributes are used to mark regions within the netlist that should not be touched when inserting test logic (including test points and fixes for bidirectional, clock, and set/reset lines).

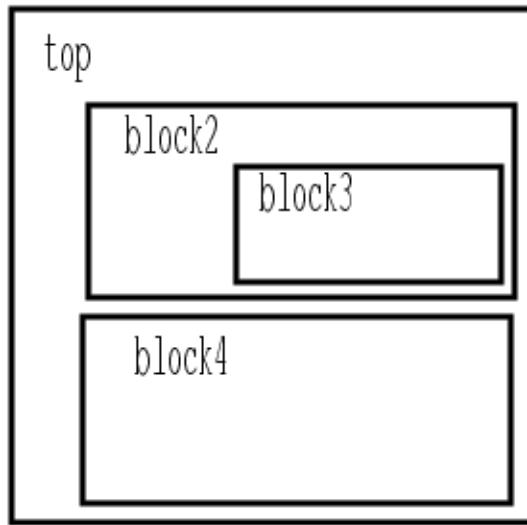
One common use model is to mark the entire netlist off-limits with the “`set_attribute_value -name no_control_point -value true`” command at the top level, and then override this by allowing test points for specific instances within the hierarchy with “`set_attribute_value /block1 -name no_control_point -value false`”.

The default attribute inheritance model works perfectly for this scenario. However, once you set the attribute for the lower level block (/block1 in this example), you will not have a way to reset to the default behavior of searching up the hierarchy. The tool will always use the specified value (true or false) for block1 and ignore the parent instances.

To get past this limitation, the inheritance model for the no_control_point and no_observe_point attributes has been extended by providing the ability to set them to true, false, or unspecified. This means you can explicitly set the value for the block (/block1 in this example) to unspecified and the tool will traverse up the hierarchy looking for an explicitly specified value.

Example 1

Figure 9-1. Hierarchical Test Logic Insertion Example



Step 1: In this example, you first set the no_control_point attribute to true for the top module. You then use the report_attributes command to get the value of the attribute at different hierarchy levels.

```
set_attribute_value [get_instances] -name no_control_point -value true  
{block1 block2 block3 ... }
```

```
SETUP> report_attributes block1
```

Attribute Definition Report

```
Attributes defined for object 'block1':
Name          Value          Inheritance
-----
...
...
...
no_control_point    true        -
no_control_reason   explicitly_specified  -
...
...
```

SETUP> report_attributes block1/block2

Attribute Definition Report

```
Attributes defined for object 'block1/block2':
Name          Value          Inheritance
-----
...
...
...
no_control_point    true        -
no_control_reason   explicitly_specified  -
...
```

Step 2: The next command allows test logic insertion inside block1 by setting the no_control_point attribute to false.

```
set_attribute_value block1-name no_control_point -value false
{block1}
```

report_attributes block1

Attribute Definition Report

```
Attributes defined for object 'block1':
Name          Value          Inheritance
-----
...
...
...
no_control_point    false       -
...
```

Example 2

Step 1: In this example, you start with the default which allows test logic insertion anywhere. You then use the report_attributes command to get the value of the no_control_point attribute at different hierarchy levels.

Step 2: The next command prevents test logic insertion inside block2 by setting the no_control_point attribute to true.

```
set_attribute_value /block1/block2 -name no_control_point -value true  
{block1/block2}
```

Step 3: Report the attribute values for block 2 and block4.

```
SETUP> report_attributes block1/block2
```

```
Attribute Definition Report
```

```
Attributes defined for object 'block1/block2':  
Name          Value          Inheritance  
-----  -----  -----  
...  
...  
...  
no_control_point      true      -  
no_control_reason    explicitly_specified  -  
...
```

```
SETUP> report_attributes block4
```

```
Attribute Definition Report
```

```
Attributes defined for object 'block4':  
Name          Value          Inheritance  
-----  -----  -----  
...  
...  
...  
no_control_point      false      -  
no_control_reason    unspecified  -  
...
```

Chapter 10

Configuration-Based Specification

This chapter documents the configuration data syntax used to encode the DftSpecification, PatternsSpecification, and DefaultsSpecification information.

The configuration data syntax is composed of nested wrappers and properties that fully describe the [DftSpecification](#), [PatternsSpecification](#), and [DefaultsSpecification/DftSpecification](#) information. These specifications can be automatically created using the [create_dft_specification](#) and [create_patterns_specification](#) commands, and they can be processed using the [process_dft_specification](#) and [process_patterns_specification](#) commands. The specifications can be edited and introspected using the commands listed in [Table 10-1](#). The specifications can also be viewed and edited in the DFTVisualizer using the [display_specification](#) command.

This chapter also includes documents how to order boundary scan cells and a description of the BISR segment order file, as described in the [MemoryBISR](#) command, that is used to order the BISR chains.

Table 10-1. Configuration Data Editing and Introspection Commands

Command	Description
add_config_element	Adds a configuration element in the configuration data.
add_config_message	Adds error, warning, or information messages to configuration elements.
add_config_tab	Adds one configuration tree tab to the DFTVisualizer Configuration Data window.
delete_config_element	Deletes one or more configuration elements. Only elements which can be added can be deleted.
delete_config_messages	Deletes error, warning or info message that were added to configuration elements using the add_config_message command.
delete_config_tabs	Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using the add_config_tab or display_specification command.
get_config_elements	Returns a collection of configuration elements or a count of configuration elements when the -count option is used.
get_config_messages	Returns a list of message strings attached to a configuration element.

Table 10-1. Configuration Data Editing and Introspection Commands (cont.)

Command	Description
<code>get_config_selected_nodes</code>	Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the Configuration Data window.
<code>get_config_value</code>	Returns a value associated with a configuration element based on the specified option.
<code>move_config_element</code>	Moves a configuration element from one location to another.
<code>read_config_data</code>	Reads the content of configuration data files or a string into the Tesson Shell environment.
<code>report_config_data</code>	Reports the content of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to a file using the <code>write_config_data</code> command.
<code>report_config_messages</code>	Reports messages associated to a configuration element. When the <code>-hierarchical</code> option is used, it also reports the messages associated to elements below the specified wrapper.
<code>report_config_syntax</code>	Reports the legal configuration syntax for a specified configuration object.
<code>set_config_value</code>	Sets the value of an element in the configuration data.
<code>write_config_data</code>	Writes the configuration data presently in memory into a file. When using the <code>-wrappers</code> option, the data written to the file can be limited to some specific wrappers.

This chapter uses the syntax conventions in [Table 10-2](#) when documenting wrappers and properties used in configuration data.

Table 10-2. SyntaxConventions for Configuration Files

Convention	Example	Usage
<i>Italic</i>	<code>scan_in : port_pin_name;</code>	An italic font indicates a user-supplied value.
Underline	<code>wgl_type : <u>generic</u> lsi;</code>	An underlined item indicates the default value.
	<code>logic_level : <u>both</u> high low;</code>	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	<code>port_naming : port_naming, ...;</code>	Ellipses indicate a repeatable value. The comment “// repeatable” also indicates a repeatable value.
//	<code>// default: ijtag_so</code>	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

Interface Naming Limitations

Ports named in Interface wrappers, as well as leaf_instance_name properties, are limited in the names they can be assigned. These names must begin with a letter and contain only alphanumerics (A-Z, a-z, 0-9) and underscores.

DftSpecification Configuration Syntax

The DftSpecification is an open and flexible interface into Tessent Shell. The interface is designed to make all Mentor Graphics DFT instruments available within Tessent Shell. In addition, the interface will eventually enable third-party providers (including end-user in-house instruments) to integrate their instrument types into the DftSpecification process. This enables chip designers to benefit from a unified yet flexible platform that supports the definition and insertion of test-related instruments from an unlimited number of sources into their designs.

This section describes the configuration data syntax used inside Tessent Shell to specify DFT components that are to be built and, optionally, inserted into a design. This chapter covers these wrappers that exist inside the DftSpecification wrapper:

- [EDT](#)
- [IjtagNetwork](#)
- [EmbeddedBoundaryScan](#)
- [BoundaryScan](#)
- [LogicBist](#)
- [MemoryBisr](#)
- [MemoryBist](#)
- [InSystemTest](#)
- [OCC](#)
- [LpctType3](#)

Using the configuration data syntax, you can specify a wide variety of DFT specifications in a single wrapper or in multiple DftSpecification wrappers, and generate and insert any type of instrument in one or multiple insertion passes.

DftSpecification	3179
EDT	3184
IjtagNetwork	3224
InSystemTest	3319
EmbeddedBoundaryScan	3339
BoundaryScan	3384
LogicBist	3445
LpctType3	3466
MemoryBisr	3478
MemoryBist	3510
OCC	3539
RtlCells	3563

DftSpecification

A wrapper that specifies the information used by the process_dft_specification command to build a series of instruments and, optionally, insert them into a design.

Usage

```
DftSpecification(module_name,id) {
    rtl_extension           : string; //default: v
    gate_extension          : string; //default: vg
    reuse_modules_when_possible : on | off | auto ;
    use_rtl_cells           : on | off | auto ;
    use_synchronizer_cell_with_reset : on | off ; //default: on *DefSpec
    use_rtl_synchronizer_cell   : on | off | auto ;
                                            //default: auto *DefSpec
    dft_cell_selection_name   : string ;
    persistent_clock_cell_prefix : string ;
                                            //default: tesson_persistent_cell_
    persistent_cell_prefix    : string ;
                                            //default: tesson_persistent_cell_
    force_creation_of_rtl_cells : cell_type, ... ; // legal : And2 Or2 Buf
                                            // Inv Mux2 ClkAnd2
                                            // ClkOr2 ClkBuf ClkInv
                                            // ClkMux2 ClkGateAnd
                                            // ClkGateOr PosedgeDff
                                            // PosedgeDffReset
                                            // PosedgeDffSet
                                            // NegedgeDff
                                            // NegedgeDffReset
                                            // NegedgeDffSet
                                            // PosedgeSynchronizerReset
                                            // PosedgeSynchronizer

    RtlCells {
    }
    BoundaryScan {
    }
    EDT {
    }
    EmbeddedBoundaryScan {
    }
    IjtagNetwork {
    }
    InSystemTest {
    }
    LogicBist {
    }
    LpctType3 {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
    OCC (id) {
    }
    //Other instrument types to be added in future releases
}
```

Description

A wrapper that specifies the information used by the [process_dft_specification](#) command to build a series of instruments and, optionally, insert them into a design.

The DftSpecification wrapper is uniquely named using two strings. The first string is the `module_name` of the design the DFT specification is for. The second string is a general purpose `id` that differentiates multiple DFT specifications for a given design module where each DftSpecification is for a different insertion pass.

Arguments

- *module_name*

A string that identifies the design that the DFT specification is for. When running the [process_dft_specification](#) command, the name of the current design must match this string unless the `-no_insertion` option is specified with this command; if the `-no_insertion` option is used, it can also be specified using the `-design_name` `design_name` option.

- *id*

A string that identifies which insertion pass the DFT specification is to be used for. When executing the [process_dft_specification](#) command, the `id` argument of the command is used to specify this `id` used in the DftSpecification name. If only one DftSpecification wrapper for a given `module_name` is loaded in memory, the [process_dft_specification](#) command uses it even if the `id` is not specified as an option. Using either “rtl” or “gate” as the `id` string is a good strategy to identify the instruments inserted in the RTL and those inserted in the synthesized netlist.

- `rtl_extension : string ;`

A string that specifies the extension used when writing out files containing Verilog RTL descriptions of the created instruments. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of `rtl_extension` must be different than the value of `gate_extension`. The default value when unspecified is “v”.

- `gate_extension : string ;`

A string that specifies the extension used when writing out files containing gate-level Verilog modules. This extension is used in the synthesis script of the instruments and when writing out the concatenated netlist. The `gate_extension` is only relevant in the `-no_rtl` context. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of `gate_extension` must be different than the value of `rtl_extension`. The default value when unspecified is “vg”.

- `reuse_modules_when_possible : on | off | auto ;`

A Boolean that specifies whether instrument modules should be reused if possible, or if different modules should be created for each instance. When the value is `auto`, it is considered to be On in the `rtl` context and Off in the `no_rtl` context. When inserting

instruments into a netlist, it is customary to avoid repeated instances of any module as timing optimization requires them to be unqualified. The default value is auto.

- use_rtl_cells : on | off | auto ;

A property that specifies whether RTL or gate-level cells are to be instantiated into the generated controllers in special locations.

For example, if a controller needs a clock multiplexer, the tool does not describe the clock multiplexer in RTL but instead instantiates a cell. When the use_rtl_cells property is specified or inferred as “on”, the tool creates the clock multiplexer as a special RTL cell and instantiates this cell into your design. You can control the name of the ports for the cell such that later you can replace the RTL cell with a real cell using the [replace_instances](#) command. Modules created by tools are often marked as “hard” to prevent accidental editing inside the modules. To replace the cell instances inside those modules, you may have to set the [is_hard_module](#) attribute to false on the parent module of the cell. Ensure that you limit the editing to only cell swapping as you do not want to make the RTL views different from their associated ICL or tcd_scan views. Refer to the [RtlCells](#) wrapper to change the port names.

When the use_rtl_cells attribute is specified or inferred as “off”, the tool uses the cells in the cell library and instantiates these cells into your design. Refer to the [get_dft_cell](#) command and the dft_cell_selection_name property to understand which specific cell the tool uses.

When the default value of “auto” is used, the on value is inferred in this specific condition:

- a. The -rtl context is used.
- b. You have not loaded a tessent cell library using the [read_cell_library](#) command containing the needed cell.

The behavior of the “auto” value was changed in the Tessent v2017.2 release to favor using real cells even in rtl context when available. Using RTL cells is no longer recommended given the lack of design module boundary preservation in the modern synthesis tool.

- use_synchronizer_cell_with_reset : on | off ;

A property that when specified to “off”, allows the tool to request a basic synchronizer cell that does not have a reset. The tool will request a synchronizer cell with reset by default when this property is specified to “on”. If the requested synchronizer cell with reset is not found in the cell library, an error is issued and the user will be prompted to set use_synchronizer_cell_with_reset to “off” to allow use of a basic synchronizer cell.

This property is currently used by the OCC, MemoryBist, and LogicBist wrappers.

- use_rtl_synchronizer_cell : on | off | auto ;

A property that when set to “on”, forces the tool to create and instantiate an RTL synchronizer cell, even when use_rtl_cells is set to “off” or when operating in the -no_rtl context. When set to “auto”, the tool will follow the use_rtl_cells behavior. In the case when the property is set to “off” and the requested synchronizer cell is not found in the cell library, an error is issued and the user is prompted to set this parameter to “on”.

This property is currently used by the OCC, MemoryBist and LogicBist wrappers.

- `force_creation_of_rtl_cells : cell_type, ... ;`

A property that specifies the creation of specific RTL cells so that you can use them in a custom insertion script such as those you would use inside the `process_dft_specification.post_insertion` procedure. You can specify the following list of RTL cells: And2, Or2, Buf, Inv, Mux2, ClkAnd2, ClkOr2, ClkBuf, ClkInv, ClkMux2, ClkGateAnd, ClkGateOr, PosedgeDff, PosedgeDffReset, PosedgeDffSet, NegedgeDff, NegedgeDffReset, NegedgeDffSet, PosedgeSynchronizer and PosedgeSynchronizerReset.

Note

 If you want to use `set_config_value` to set this property with a list of cell names, use a Tcl list of cell names. The list of RTL cells will result in a comma-separated list in the configuration data. See “[Example 2](#)” on page 1995 in the `set_config_value` command description.

- `dft_cell_selection_name : string ;`

A property that specifies the name of the `dft_cell_selection` wrapper from which the cells are selected. You only need to specify this property if all of the following apply:

- RTL cells are not used
- Your cell library has more than one `dft_cell_selection` wrapper
- You have not specified a default `dft_cell_selection` wrapper using the `set_cell_library_options -default_dft_cell_selection_name` command

- `persistent_clock_cell_prefix : string ; // default: tessent_persistent_cell_`

A property that specifies the leaf instance name prefix to use when instantiating a clock cell such as a clock multiplexer or a clock gating cell. The string must start with a letter and be followed by only letters, numbers, and underscores. When unspecified, it defaults to “`tessent_persistent_cell_`”. These clock cells have a special prefix so that they are not removed during synthesis and layout. You can modify this string to match the prefix you normally use in your design flow; otherwise, use the following Design Compiler command in your synthesis and layout scripts:

```
set_size_only -all_instance \
  [get_cells tessent_persistent_cell_* -hierarchical \
  -filter "is_hierarchical==false"]
```

Refer to the SDC proc described in the “[Generating and Using SDC for Tessent Shell Embedded Test IP](#)” in the *Tessent Shell User's Manual* for more information. Specifically, look at the Synthesis Step 3 in the “[Example Design Compiler Synthesis Script](#)”.

- `persistent_cell_prefix : string ; // default: tessent_persistent_cell_`

A property that specifies a leaf instance name prefix to use when instantiating a non-clock cell such as a buffer or an inverter. The string must start with a letter and be followed by only letters, numbers, and underscores. When unspecified, it defaults to

“tesson_t_persistent_cell_”. These non-clock cells have a special prefix to ensure that they are not removed during synthesis and layout so that they can be used as anchor points for timing constraints or as start and end points in ScanDef. You can modify this string to match the prefix you normally use in your design flow; otherwise, use the following Design Compiler command in your synthesis and layout scripts:

```
set_size_only -all_instance \
[get_cells tesson_t_persistent_cell_* -hierarchical \
-filter "is_hierarchical=false"]
```

Refer to the SDC proc described in the “[Generating and Using SDC for Tessent Shell Embedded Test IP](#)” in the *Tessent Shell User’s Manual* for more information. Specifically, look at the Synthesis Step 3 in the “[Example Design Compiler Synthesis Script](#)”.

EDT

A wrapper that specifies the information used by the [process_dft_specification](#) command to build the EDT IP and, optionally, insert the IP into a design.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        ijtag_host_interface : ijtag_leaf_node_name | none;
        Connections {
        }
        Controller(id) {
        }
    }
}
```

Description

A wrapper that specifies the information used by the [process_dft_specification](#) command to build the EDT IP and, optionally, insert the IP into a design.

Arguments

- `ijtag_host_interface : ijtag_leaf_node_name | none;`

This optional property is used to specify the host scan interface that the EDT controllers will connect to. This property has an override inside the [Controller](#) wrapper.

The *node_id* value must have the format Sib(*id*), Tap(*id*)/HostIjtag(*id*), HostScanInterface(*id*), ScanMux(*id*)/Input(0|1) and must match the leafname of a wrapper in the [IjtagNetwork](#) wrapper.

To have the IjtagNetwork created with a Sib node for the service of the EDT node, use the `-sri_sib_list` switch when calling the [create_dft_specification](#) command. For example, use `-sri_sib_list {edt occ}` when inserting the logic test hardware.

When you specify `ijtag_host_interface` with a value pointing to an IJTAG host scan interface during `process_dft_specification`, a TDR is added that drives the static external controls of the EDT. If you have also specified static external controls which are tied to top level ports or pins, the TDR connections to the EDT static external controls override the specified static external controls.

When the property is unspecified, it defaults to none which may be the inherited values by all controller wrappers if the `ijtag_host_interface` property is not specified locally.

Connections

This wrapper is used to specify EDT control pin connections for each EDT controller.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Connections {
            edt_clock      : port_pin_name | DftSignal(edt_clock) |
                               OptionalDftSignal(edt_clock);
            edt_slave_clock : port_pin_name; // default: edt_slave_clock
            edt_update      : port_pin_name | DftSignal(edt_update) |
                               OptionalDftSignal(edt_update);
            edt_reset       : port_pin_name; // default: edt_reset
            StaticExternalControls {
            }
        }
    }
}
```

Description

This wrapper is used to specify EDT control pin connections for each EDT controller.

Arguments

- `edt_clock : port_pin_name ;`

An optional property used to specify a source node to connect the `edt_clock` port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the `set_design_level` command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(edt_clock)` or `DftSignal(edt_clock)` as the value, and it will be interpreted to request a connection to the DFT signal `edt_clock` as it was added using the `add_dft_signals` command. When using `OptionalDftSignal(edt_clock)`, you will not get an error if the DFT signal `edt_clock` was not added; a connection to the port called “`edt_clock`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value defaults to `OptionalDftSignal(edt_clock)`.

- `edt_slave_clock : port_pin_name ;`

An optional property that specifies the EDT slave clock. When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. See “[TestKompress Compression Logic](#)” in the *Tessent TestKompress User’s Manual*. The default is `edt_slave_clock`.

- `edt_update : port_pin_name ;`

An optional property used to specify a source node to connect the `edt_update` port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(edt_update)` or `DftSignal(edt_update)` as the value, and it will be interpreted to request a connection to the DFT signal `edt_update` as it was added using the [add_dft_signals](#) command. When using `OptionalDftSignal(edt_update)`, you will not get an error if the DFT signal `edt_update` was not added; a connection to the port called “`edt_update`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value defaults to `OptionalDftSignal(edt_update)`.

- `edt_reset : port_pin_name ;`

An optional property that specifies the port or pin name to be used for the EDT reset. The default is `edt_reset`. The `edt_reset` signal is an asynchronous reset for all the sequential elements in the EDT logic. For more information, see “[EDT Logic Reset](#)” in the *Tessent TestKompress User’s Manual*.

StaticExternalControls

This wrapper specifies the connections for the static EDT control signals.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Connections {
            StaticExternalControls {
                edt_bypass : port_pin_name | 0 | 1 ; //default: 0
                edt_single_bypass_chain : port_pin_name | 0 | 1 ; //default: 0
                edt_configuration : port_pin_name | 0 | 1 ; //default: 0
                edt_low_power_shift_enable : port_pin_name | 0 | 1 ; //default: 1
            }
        }
    }
}
```

Description

This wrapper specifies the connections for the static EDT control signals.

Arguments

- **edt_bypass : *port_pin_name* ;**
An optional property that specifies the connection to the *edt_bypass* static EDT control pin. The *edt_bypass* pin connects to the specified *port_pin_name*, or is tied to high or low using the 1 or 0 symbol. The default is 0.
- **edt_single_bypass_chain : *port_pin_name* ;**
An optional property that specifies the connection to the *edt_single_bypass_chain* static EDT control pin. The *edt_single_bypass_chain* pin connects to the specified *port_pin_name*, or is tied to high or low using the 1 or 0 symbol. The default is 0.
- **edt_configuration : *port_pin_name* ;**
An optional property that specifies the connection to the *edt_configuration* static EDT control pin. The *edt_configuration* pin connects to the specified *port_pin_name*, or is tied to high or low using the 1 or 0 symbol. The default is 0.
- **edt_low_power_shift_enable : *port_pin_name* ;**
An optional property that specifies the connection to the *edt_low_power_shift_enable* static EDT control pin. The *edt_low_power_shift_enable* pin connects to the specified *port_pin_name*, or is tied to high or low using the 1 or 0 symbol. The default is 1.

Controller

This wrapper is used to add and specify properties for every EDT controller generated with `process_dft_specification`.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            ijttag_host_interface      : ijttag_leaf_node_name | none;
            longest_chain_range       : min, max ;
            scan_chain_count          : int ;
            input_channel_count       : int ;
            output_channel_count      : int ;
            separate_control_data_channels : on | off ;
            parent_instance            : name ;
            leaf_instance_name         : name ;
            connect_bscan_segments_to_lsb_chains : on | off | auto ;
            Interface {
            }
            BypassChains {
            }
            Compactor {
            }
            Clocking {
            }
            HighCompressionConfiguration {
            }
            ShiftPowerOptions {
            }
            LogicBistOptions {
            }
            Connections {
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for every EDT controller generated with `process_dft_specification`.

Every Controller needs to have an unique simple name. You must use a name that starts with an alphanumeric character and contains no special characters except for underscores "_" in the middle of the controller name. You can also use an integer to specify Controller names. For example, the following controller names are valid:

```
Controller(1)
Controller(c1)
Controller(controller_1)
```

Arguments

- `ijtag_host_interface : ijtag_leaf_node_name | none;`

An optional property that connects the EDT static external controls through a test data register (TDR). The `ijtag_host_interface` is inherited from the top-level. If it is specified at this level, it overrides the `ijtag_host_interface` specified at the top-level.

The `node_id` value must have the format `Sib(id)`, `Tap(id)/HostIjtag(id)`, `HostScanInterface(id)`, `ScanMux(id)/Input(0|1)` and must match the leafname of a wrapper in the [IjtagNetwork](#) wrapper.

To have the `IjtagNetwork` created with a `Sib` node for the service of the EDT node, use the `-sri_sib_list` switch when calling the [create_dft_specification](#) command. For example, use `-sri_sib_list {edt occ}` when inserting the logic test hardware.

When you specify `ijtag_host_interface` with a value pointing to an IJTAG host scan interface during `process_dft_specification`, a TDR is added that drives the static external controls of the EDT. If you have also specified static external controls which are tied to top level ports or pins, the TDR connections to the EDT static external controls override the specified static external controls.

When the property is unspecified, it inherits its value from the `ijtag_host_interface` property found in the [EDT](#) wrapper.

- `longest_chain_range : min, max ;`

A required property, for each controller, that specifies the range (min and max values) for the length of the longest scan chain. Use this switch to avoid having to regenerate the logic for minor changes.

- `scan_chain_count : int ;`

A required property, for each controller, that specifies the number of scan chains for the EDT controller. It provides the size for the scan chain bus ports. The tool will report an error if this property is missing or has no value.

- `input_channel_count : int ;`

A required property, for each controller, that specifies the number of EDT input channels in the generated EDT controller. It provides the size for the input channel bus port. The tool will report an error if this property is missing or has no value.

- `output_channel_count : int ;`

A required property, for each controller, that specifies the number of EDT output channels in the generated EDT controller. It provides the size for the output channel bus port.

- `separate_control_data_channels : on | off ;`

An optional property that specifies if control data channel separation is used in the generated EDT controller. The default is off. When set to on, the tool automatically determines the number of control channels and inserts control bits behind the control channels only. This is necessary for channel sharing between non-identical blocks.

- `parent_instance : name ;`

An optional property that specifies where in the design to insert the EDT controller. If not specified, the controller instance is created at the top level of the current design.

- `leaf_instance_name : name ;`

An optional property that specifies the EDT controller instance name. If not specified, the name is “`<design_name>_<dft_spec_id>_tessent_<edt_id>_inst`”. The `dft_spec_id` is the id specified for the DftSpecification wrapper and `edt_id` is the id specified for the EDT Controller wrapper.

- `connect_bscan_segments_to_lsb_chains : on | off | auto;`

An optional property that specifies whether to connect boundary scan chain segments to the EDT IP automatically if boundary scan is inserted in the same pass or a previous insertion pass. When this property is set to auto and a single EDT controller is present in the configuration data, then the insertion phase will connect the boundary scan chain segments to the EDT controller.

When boundary scan chain segments are created, the boundary scan instrument dictionary is populated with information about them. When you want to connect the EDT controller to boundary scan chain segments, the insertion phase collects boundary scan chain segment information from the instrument dictionary and uses the data to perform connections. For more information on the instrument dictionary see [get_instrument_dictionary](#).

The tool will not allow more than one EDT controller to connect to boundary scan chain segments. When there are more than one EDT controllers present, you need to specify which EDT controller to connect to the boundary scan chain segments. You accomplish this by setting this property to on for the EDT controller that you want to connect. When all EDT controllers have this property set to auto, or when more than one EDT controller have this property set to on, an error is reported during the validation phase.

The width of the EDT scan input and output bus on the EDT controller module is set with the `scan_chain_count` property value. If boundary scan chain segments exist and are to be connected to the EDT controller, the width of the bus is incremented by the number of boundary scan segments.

For example, if you have 100 scan chains and 5 boundary scan chain segments, the resulting bus width on the EDT controller is 105 bits. This is so there are enough bits when connecting scan chains from scan insertion.

As suggested by the name for this property, `connect_bscan_segments_to_lsb_chains`, the boundary scan chain segments are connected to the LSB bits of the EDT scan bus. Each boundary scan chain segment connection is made to the corresponding bit of the bus, so the first segment is connected to the first bit, and so on.

The default value for this property is auto.

Examples

Example 1

This example sets connect_bscan_segments_to_lsb_chains to auto, which results in the EDT controller being connected to the boundary scan chain segments.

```
DftSpecification(CoreA,rtl) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Tap(1) {
                HostBscan {}
            }
        }
    }
    BoundaryScan {
        ijtag_host_node : Tap(1)/HostBscan;
        max_segment_length_for_logictest : 10;
    }
    EDT {
        Controller(c1) {
            longest_chain_range : 100, 110;
            scan_chain_count : 100;
            input_channel_count : 2;
            output_channel_count : 2;
            connect_bscan_segments_to_lsb_chains : auto;
        }
    }
}
```

Interface

This wrapper specifies the port names for the interface of the EDT controller.

Usage

```
DftSpecification(module_name,id) {
    EDT {
        Controller(id) {
            Interface {
                edt_clock           : port_name ; // default: edt_clock
                edt_slave_clock     : port_name ; // default: edt slave clock
                edt_update          : port_name ; // default: edt update
                edt_reset           : port_name ; // default: edt reset
                edt_channels_in_bus : bus_name   ; // default: edt channels in
                edt_channels_out_bus: bus_name   ; // default: edt channels out
                StaticExternalControls {
                }
            }
        }
    }
}
```

Description

You can use this wrapper to specify the names of the EDT controller ports. Each property has a default value, as described in the configuration data syntax. These port names are for non-static EDT signals. The [StaticExternalControls](#) wrapper specifies static EDT signal port names for the EDT controller.

Note

 Instance names inside the Interface wrapper must start with a letter and consist only of alphanumerics and underscores.

Arguments

- `edt_clock : port_name ;`

An optional property that specifies the name of the port that supplies a clock to the EDT controller. The default is `edt_clock`. See the `edt_clock` entry in [Table 3-11](#).

- `edt_slave_clock : port_pin_name ;`

An optional property that specifies the name of the EDT controller slave clock port. When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. The default is `edt_slave_clock`. See “[TestKompress Compression Logic](#)” in the *Tessent TestKompress User’s Manual*.

- `edt_update : port_name ;`

An optional property that specifies name of the update enable port of the EDT controller. The default is `edt_update`. See the `edt_update` entry in [Table 3-11](#).

- `edt_reset : port_name ;`
An optional property that specifies the name of the asynchronous reset port for the EDT controller. This property applies when the controller clocking reset signal is asynchronous. The default is `edt_reset`.
- `edt_channels_in_bus : bus_name ;`
An optional property that specifies the name of the input channels bus port for the EDT controller. The default is `edt_channels_in`.
- `edt_channels_out_bus : bus_name ;`
An optional property that specifies the name of the output channels bus port for the EDT controller. The default is `edt_channels_out`.

StaticExternalControls

This wrapper specifies the port names for the static EDT signals on the interface of the EDT controller.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Interface {
                StaticExternalControls {
                    edt_bypass : port_name ;
                    // default: edt bypass
                    edt_single_bypass_chain : port_name ;
                    // default: edt single bypass chain
                    edt_configuration : port_name ;
                    // default: edt configuration
                    edt_low_power_shift_enable : port_name ;
                    // default: edt low power shift en
                }
            }
        }
    }
}
```

Description

This wrapper specifies the names of static EDT signals.

Arguments

- `edt_bypass : port_name ;`
An optional property that specifies the name of the bypass port for the EDT controller. The default is `edt_bypass`.
- `edt_single_bypass_chain : port_name ;`
An optional property that specifies the name of the single bypass chain port for the EDT controller. The default is `edt_single_bypass_chain`.
- `edt_configuration : port_name ;`
An optional property that specifies the EDT configuration port for the EDT controller. The default is `edt_configuration`.
- `edt_low_power_shift_enable : port_name ;`
An optional property that specifies low-power shift enable port for the EDT controller. The default is `edt_low_power_shift_enable`.

BypassChains

This wrapper is used to specify the EDT bypass chain configuration in the generated EDT controller.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            BypassChains {
                present           : on | off ;
                bypass_chain_count : int ; // default: auto
                single_bypass_chain : on | off | auto ;
                BypassChain(id) { // legal: 1..maxposint
                    scan_chain_range_list : scan_chain_range, ... ;
                }
            }
        }
    }
}
```

Description

This wrapper is used to specify the EDT bypass chain configuration in the generated EDT controller.

Arguments

- **present : on | off ;**
An optional property that specifies whether the bypass chain is present. The default is on.
- **bypass_chain_count : *int* ;**
An optional property that specifies the bypass chain count. The default (auto) is the input or output channel count, whichever is smaller. The default is auto.
- **single_bypass_chain : on | off | auto ;**
An optional property that specifies whether a single bypass chain is present. The default is auto, with the following behavior:
 - If a LogicBist wrapper is not specified, this property switches to off.
 - If a LogicBist wrapper is specified, this property takes the value of the LogicBist/Controller/[SingleChainForDiagnosis](#)/present property.
- **BypassChain(*id*)**
A repeatable wrapper that specifies the bypass chain connections. The *id* value for this wrapper is used to specify the actual bypass chain index. You must specify the index within the bypass chain count limits. You cannot specify twice the same value for a single EDT controller. The *scan_chain_range_list* property inside this wrapper is used to specify the scan chains that create a specific bypass chain.

- `BypassChain(id)/scan_chain_range_list : scan_chain_range, ... ;`

A property that specifies a list of scan chains by their index value.

The syntax allows to use a list of comma separated range values. The format for the ranges is “*int:int*”. The order of the specified scan chains has impact on the order of scan chain connections in the generated EDT IP. The specified scan chain index values must fall within the limits specified by the `scan_chain_count` property of the EDT controller.

It is also important to remember that a single scan chain can be assigned to its connection only once and that all scan chains need to have a connection assigned. Otherwise, a validation error is reported.

Examples

Example 1

This example shows the BypassChains wrapper with the chain connections specified:

```
DftSpecification(CoreA,rtl) {
    EDT {
        Controller(c1) {
            scan_chain_count : 16;
            input_channel_count : 2;
            output_channel_count : 2;
            longest_chain_range : 100, 110;
            BypassChains {
                present: on;
                bypass_chain_count: 2;
                BypassChain(1) {
                    scan_chain_range_list: 1:2, 4:10;
                }
                BypassChain(2) {
                    scan_chain_range_list: 3, 11:16;
                }
            }
        }
    }
}
```

Compactor

This wrapper is used to add and specify properties for the EDT compactor.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id)
        Compactor {
            type : xpress | basic ;
            pipeline_logic_levels_in_compactor : int ; // default: off
            CompactorConnection(id) { // legal: 1..maxposint
                scan_chain_range_list : scan_chain_range, ... ;
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT compactor.

Arguments

- `type : enum ;`

An optional property that determines which compactor is used in EDT logic. This is equivalent to the “-COMpactor_type” argument in the [set_edt_options](#) command. The possible values are:

- **xpress** — The Xpress compactor, which increases compression, especially for designs that generate unknown (X) values. This is the default value.
- **basic** — The basic compactor is the first-generation compactor.

For more information, refer to “[Understanding Compactor Options](#)” in the *Tessent TestKompress User’s Manual*.

- `pipeline_logic_levels_in_compactor : int ;`

An optional property that enables pipeline stages. This is equivalent to the “-Pipeline_logic_levels_in_compactor” argument in the [set_edt_options](#) command. The possible values are:

- **off** — Disables pipeline stages in EDT logic. This is the default value.
- **<integer>** — Includes pipeline stages in the EDT logic compactor(s) and specifies the maximum number of logic levels (2-input XOR gates) allowed in the compactors before inserting a pipeline stage.

- `CompactorConnection(id)`

A repeatable wrapper that specifies compactor connections.

The *id* value of this wrapper is used to specify the actual EDT output channel index. You must specify the index within the EDT output channel count limits. You cannot specify the same value twice for a single EDT controller. The *scan_chain_range_list* property inside this wrapper is used to specify the scan chains that will be connected to the EDT output channel.

- CompactorConnection(*id*)/*scan_chain_range_list* : *scan_chain_range*, ... ;

A required property for each CompactorConnection that specifies a list of scan chains by their index values. The syntax for the property value is a list of comma-separated range values. The ranges are of the format “*int:int*”. The order of the specified scan chains impacts the order of scan chain connections in the generated EDT IP. The specified scan chain index values must be within the limits specified by the *scan_chain_count* property of the EDT controller.

Note



A single scan chain can be assigned to its connection only once, and all scan chains need to have a connection assigned. Otherwise, a validation error is reported.

Examples

Example 1

This example shows the specification wrapper with the compactor connections specified:

```
DftSpecification(CoreA,rtl) {
    EDT {
        Controller(c2) {
            scan_chain_count : 16;
            input_channel_count : 2;
            output_channel_count : 2;
            longest_chain_range : 100, 110;
            Compactor {
                type: xpress;
                CompactorConnection(1) {
                    scan_chain_range_list: 1:2, 4:10;
                }
                CompactorConnection(2) {
                    scan_chain_range_list: 3, 11:16;
                }
            }
        }
    }
}
```

Clocking

A wrapper that specifies the information used by the EDT controller clock.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Clocking {
                type      : edge | level ;
                lockup_cells : on | off ;
                reset_signal : asynchronous | off ;
            }
        }
    }
}
```

Description

A wrapper that specifies the information used by the EDT controller clock.

Arguments

- type : edge | level ;

An optional property that determines what type of architecture logic is used for the EDT logic. The default is edge. For more information, refer to the -CLOCKING argument of the [set_edt_options](#) command.

- lockup_cells : on | off ;

An optional property that determines whether lockup cells are inserted in the EDT logic. The default is on. For more information, refer to the -LOCKUP_cells argument of the [set_edt_options](#) command.

Caution

 It is recommended to leave this property set to on. Disabling it can lead to DRC violations during pattern generation and the inability to generate valid patterns.

- reset_signal : asynchronous | off ;

A property that specifies whether an asynchronous reset signal, `edt_reset`, is included in the EDT logic. For more information refer to the -RESET_SIGNAL argument of the [set_edt_options](#) command.

HighCompressionConfiguration

This wrapper is used to specify high-compression properties for the EDT controller.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            HighCompressionConfiguration {
                present : on | off ;
                input_channel_count : int ;
                output_channel_count : int ;
            }
        }
    }
}
```

Description

Use this wrapper to specify the presence of the high-compression configuration for the EDT controller and the input and output channel counts for the high-compression configuration. For more information, see “[Dual Compression Configurations](#)” in the *Tessent TestKompress User’s Manual*.

Arguments

- **present : on | off ;**
A property that, when set to on, specifies that the high-compression EDT configuration is available for the EDT controller. When this property is set to on, the -parameter switch of the [add_core_instances](#) command allows a `high_compression` or `low_compression` value for the `edt_configuration` core instance parameter. The default is off.
- **input_channel_count : int ;**
A property that specifies the input channel count for the high compression configuration of the EDT controller.
- **output_channel_count : int ;**
A property that specifies the output channel count for the high compression configuration of the EDT controller.

Examples

This example shows the DftSpecification of an EDT controller. It uses a HighCompressionConfiguration wrapper, highlighted in green. The specified settings result in the generation of a second EDT configuration with one input channel and one output channel. This is in addition to the low compression configuration defined in the Controller wrapper.

```
DftSpecification(CoreA,rtl) {
    EDT {
        Controller(c1) +{
            scan_chain_count : 16;
            input_channel_count : 2;
            output_channel_count : 2;
            longest_chain_range : 100, 110;
            BypassChains {
                present: on;
                single_bypass_chain: on;
            }
            HighCompressionConfiguration {
                present : on ;
                input_channel_count : 1 ;
                output_channel_count : 1 ;
            }
            ShiftPowerOptions {
                present : on ;
            }
            Connections +{
                StaticExternalControls {
                    edt_bypass: top_edt_bypass;
                    edt_single_bypass_chain: top_edt_single_bypass_chain;
                    edt_configuration: top_edt_configuration;
                    edt_low_power_shift_enable: top_edt_low_power_shift_enable;
                }
            }
        }
    }
}
```

ShiftPowerOptions

This wrapper is used to add and specify properties for the EDT shift power options.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            ShiftPowerOptions {
                present : on | off ;
                full_control : on | off ;
                min_switching_threshold_percentage : int ; // default: 15
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT shift power options.

Arguments

- **present : on | off ;**
A property used to specify if the EDT low-power controller is present. The default is off.
- **full_control : on | off ;**
A property and literal value that adds one bit per scan chain to control whether that chain loads a constant value or not. This provides complete control over the scan chain input masking. When this property is on, the tool adds one bit per scan chain. Also, when on, any value specified with the -min_switching_threshold_property is ignored. The default is off.
- **min_switching_threshold_percentage : *int* ;**
A property and integer that specifies the minimum switching threshold value used to determine power controller parameters, such as input shift register size. Valid integer values are 1 to 50. If an integer greater than 50 is specified, the tool uses 50. The default value is 15, indicating 15%.

The tool uses this value to calculate the size of the power_controller. The power controller must be large enough to control the number of scan chains required to achieve the minimum switching threshold specified. For more information, see “[Low-Power Shift and Switching Thresholds](#)” in the *Tessent TestKompress User’s Manual*.

LogicBistOptions

This wrapper groups the LogicBist-related options set at the EDT controller level.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            LogicBistOptions {
                present : on | off | auto ;
                misr_input_ratio : integer ;           // default: auto
                chain_mask_register_ratio : integer ;   // default: 1
                ShiftPowerOptions {
                }
            }
        }
    }
}
```

Description

You use this wrapper to set LogicBIST-related options for the EDT controller.

Arguments

- present : on | off | auto ;

A property that specifies the generated EDT controller will be a hybrid TK/LBIST controller. The default, auto, indicates that when the DftSpecification/LogicBist wrapper is used, this property will be set to on and hybrid controllers are generated.

- misr_input_ratio : *integer* ;

A property and integer pair that specifies the ratio of the number of scan chains to the number of MISR inputs. The default is auto. For more information, see the lbist_misr_input_ratio argument description for the [set_edt_options](#) command. A value of auto is equivalent to not specifying the lbist_misr_input_ratio argument for the [set_edt_options](#) command.

- chain_mask_register_ratio : *integer* ;

A property and integer pair that specifies the number of scan chains controlled by a single EDT chain mask register bit. By default, the ratio is 1:1; there are as many bits in the chain mask register as there are number of scan chains.

You can use this property to minimize the area impact of the chain mask register. For example, specifying 3 for *integer* means a 3:1 ratio with each chain mask register bit controlling 3 scan chains. Specify the ratio on a per EDT block basis.

The larger the ratio you specify, the more you trade off resolution for area impact. When any scan chain connected to a chain mask register bit is masked, all the other scan chains connected to the bit are also masked automatically.

Refer to “[Scan Chain Masking](#)” in the *Hybrid TK/LBIST User’s Manual* for more information.

ShiftPowerOptions

This wrapper groups properties related to Low Power Shift capabilities.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            LogicBistOptions {
                ShiftPowerOptions {
                    present : off | on ;
                    default_operation : enabled | disabled ;
                    SwitchingThresholdPercentage{
                    }
                }
            }
        }
    }
}
```

Description

This wrapper allows you to set low power shift options for the hybrid TK/LBIST controller.

Arguments

- **present : on | off**
A property which enables the low power shift functionality for the hybrid TK/LBIST controller.
- **default_operation : enabled | disabled**
A property which specifies whether low power shift should be enabled or disabled in the default hardware mode.

SwitchingThresholdPercentage

This wrapper allows you to set the switching threshold percentage for the hybrid TK/LBIST controller.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            LogicBistOptions {
                ShiftPowerOptions {
                    SwitchingThresholdPercentage {
                        min : integer ; //default: 15
                    }
                }
            }
        }
    }
}
```

Arguments

- *min* : *integer* ;

An optional switch and integer pair that specifies the switching threshold value for the hardware mode for the Hybrid TK/LBIST controller. Valid Values are between 1 and 50. The default is 15. For more information, see the [set_lbist_power_controller_options](#) command.

Connections

This wrapper is used to add and specify properties for the EDT controller connections.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                edt_clock      : port_pin_name | DftSignal(edt_clock) | OptionalDftSignal(edt_clock);
                edt_slave_clock : port_pin_name;
                edt_update      : port_pin_name | DftSignal(edt_update) | OptionalDftSignal(edt_update);
                edt_reset       : port_pin_name;
                StaticExternalControls {
                }
                EdtChannelsIn(id) {
                }
                EdtChannelsOut(id) {
                }
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT controller connections.

There are cases when you may need to connect multiple EDT channel connections to a single top level port. A method for doing this, along with an example, is described in the documentation for the [add_dft_modal_connections](#) command.

Arguments

- `edt_clock : port_pin_name | DftSignal(edt_clock) | OptionalDftSignal(edt_clock);`
An optional property used to specify a source node to connect the `edt_clock` port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(edt_clock)` or `DftSignal(edt_clock)` as the value, and it will be interpreted to request a connection to the DFT signal `edt_clock` as it was added using the [add_dft_signals](#) command. When using `OptionalDftSignal(edt_clock)`, you will not get an error if the DFT signal `edt_clock` was not added; a connection to the port called “`edt_clock`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value is inherited from the ..[/Connections](#)/edt_clock property which itself defaults to OptionalDftSignal(edt_clock) when unspecified.

- edt_slave_clock : *port_pin_name* ;

A property that specifies the EDT slave clock. When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. See “[TestKompress Compression Logic](#)” in the *Tessent TestKompress User’s Manual*. The default is edt_slave_clock.

- edt_update : *port_pin_name* | DftSignal(edt_update) | [OptionalDftSignal\(edt_update\)](#);

An optional property used to specify a source node to connect the edt_update port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_update) or DftSignal(edt_update) as the value, and it will be interpreted to request a connection to the DFT signal edt_update as it was added using the [add_dft_signals](#) command. When using OptionalDftSignal(edt_update), you will not get an error if the DFT signal edt_update was not added; a connection to the port called “edt_update” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value is inherited from the ..[/Connections](#)/edt_update property which itself defaults to OptionalDftSignal(edt_update) when unspecified.

- edt_reset : *port_pin_name* ;

A property that specifies edt_reset. The default is edt_reset.

StaticExternalControls

This wrapper is used to add and specify properties for the EDT static external controls.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                StaticExternalControls {
                    edt_bypass : port_pin_name | 0 | 1 ;
                    edt_single_bypass_chain : port_pin_name | 0 | 1 ;
                    edt_configuration : port_pin_name | 0 | 1 ;
                    edt_low_power_shift_enable : port_pin_name | 0 | 1 ;
                }
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT static external controls.

Arguments

- `edt_bypass : port_pin_name | 0 | 1 ;`

A property that specifies the connection to the `edt_bypass` EDT static control pin. The `edt_bypass` pin connects to the specified `port_pin_name`, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the `EDT/Connections/StaticExternalControls/edt_bypass` specification. The default is 0.

- `edt_single_bypass_chain : port_pin_name | 0 | 1 ;`

A property that specifies the connection to the `edt_single_bypass_chain` static EDT control pin. The `edt_single_bypass_chain` pin connects to the specified `port_pin_name`, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the `EDT/Connections/StaticExternalControls/edt_single_bypass_chain` specification. The default is 0.

- `edt_configuration : port_pin_name | 0 | 1 ;`

A property that specifies the connection to the `edt_configuration` static EDT control pin. The `edt_configuration` pin connects to the specified `port_pin_name`, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the `EDT/Connections/StaticExternalControls/edt_configuration` specification. The default is 0.

- `edt_low_power_shift_enable : port_pin_name | 0 | 1 ;`

A property that specifies the connection to the `edt_low_power_shift_enable` static EDT control pin. The `edt_low_power_shift_enable` pin connects to the specified `port_pin_name`, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the `EDT/Connections/StaticExternalControls/edt_low_power_shift_enable` specification. The default is 1.

EdtChannelsIn

This wrapper is used to add and specify properties for the EDT channel in ports.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                EdtChannelsIn(id_range) {
                    port_pin_name : port_pin_name ;
                    pipeline_clock : port_pin_name ; // default : edt_clock
                    PipelineStage {
                    }
                }
            }
        }
    }
}
```

Description

The `process_dft_specification` command for EDT creates bus ports for EDT channel connections on the block level as specified in the configuration data. If the bus ports exist already, they are used for EDT channel connections. If they do not exist, they will be created on the block level by `process_dft_specification`.

You specify the bus names with the `port_pin_name` properties with this configuration data wrapper. The ports must have sufficient bits to make all of the specified EDT channel input connections. EDT `process_dft_specification` cannot enhance existing ports with additional bits.

You can specify the bus ports to be created with any bit range as long as it is a valid Verilog range. They can start with 0 or any other number. Any range is acceptable as long as the EDT channel range width matches the specified connection width. The order you specify the bus ports has an impact on the connection order made to the EDT IP.

The (*id*) component of the `EdtChannelsIn` wrapper accepts a range of positive integer values using a single value or a colon-separated pair of values. For example:

```
EdtChannelsIn(4:1) - first 4 EDT input channels
EdtChannelsIn(1:4) - first 4 EDT input channels with inverted connection
                     order
EdtChannelsIn(1)   - first EDT input channel
EdtChannelsIn(1:1) - first EDT input channel
```

When you specify at least one input connection, you must specify all values and ranges.

Note

 The EDT `process_dft_specification` does not expand existing bus ports. An error will be reported if you use more bits than are on the existing bus.

Arguments

- `port_pin_name : port_pin_name ;`

A property that specifies signals that are connected to the `edt_channels_in` port of the EDT controller. The value can be any of these:

- bus
- list of scalar ports/pins
- list of mixed bus and port/pin objects
- indexed identifier

- `pipeline_clock : port_pin_name ;`

An optional property that specifies the pipeline clock. If not specified the default is `edt_clock`. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Input Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Examples

Example 1

In this example, the tool connects the first four EDT input channels to the `top_bus[3:0]` port object. The first EDT channel connects to `top_bus[0]` and the fourth channel connects to `top_bus[3]`.

```
EdtChannelsIn(4:1) {
    port_pin_name : top_bus[3:0];
}
```

Example 2

This example demonstrates using a mixed list of scalar ports and buses.

```
EdtChannelsIn(8:5) {
    port_pin_name : top8, top7, in[1:0];
}
```

The tool makes connections as follows:

```
edt_ctrl_inst/edt_channels_in[7] -> top8
edt_ctrl_inst/edt_channels_in[6] -> top7
edt_ctrl_inst/edt_channels_in[5] -> in[1]
edt_ctrl_inst/edt_channels_in[4] -> in[0]
```

Example 3

This example uses an indexed bus identifier.

```
EdtChannelsIn(4:1) {
    port_pin_name : in[%d];
}
```

The tool automatically infers the bus bit values for the connection as follows:

```
edt_ctrl_inst/edt_channels_in[3] -> in[3]
edt_ctrl_inst/edt_channels_in[2] -> in[2]
edt_ctrl_inst/edt_channels_in[1] -> in[1]
edt_ctrl_inst/edt_channels_in[0] -> in[0]
```

Example 4

This example shows how to create two EDT controllers. While it shows the creation of both input and output bus ports, notice the effect of EdtChannelsIn.

```
DftSpecification(CoreB,rtl) {
    EDT {
        Controller(1) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst1;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[3:0];
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[3:0];
                }
            }
        }
        Controller(2) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst2;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[15:12];
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[15:12];
                }
            }
        }
    }
}
```

With this configuration data in place, two new EDT controllers will be created when you run the process_dft_specification command, edt_ctrl_inst1 and edt_ctrl_inst2. The input EDT channels

in each controller are connected to the created bus ports named “new_input_bus” by the EdtChannelsIn wrapper. This shows the connections made for both controllers:

```
edt_ctrl_inst1/edt_channels_in[3] -> new_input_bus[3]
edt_ctrl_inst1/edt_channels_in[2] -> new_input_bus[2]
edt_ctrl_inst1/edt_channels_in[1] -> new_input_bus[1]
edt_ctrl_inst1/edt_channels_in[0] -> new_input_bus[0]
edt_ctrl_inst2/edt_channels_in[3] -> new_input_bus[15]
edt_ctrl_inst2/edt_channels_in[2] -> new_input_bus[14]
edt_ctrl_inst2/edt_channels_in[1] -> new_input_bus[13]
edt_ctrl_inst2/edt_channels_in[0] -> new_input_bus[12]
```

The new bus ports will have a range of 16 bits (15:0) each because the MSB specified is 15 and the LSB is 0 for both of them. The tool will always create every bus port from its MSB to LSB bits as specified in the configuration data.

PipelineStage

This wrapper is used to add and specify properties for the EDT channel in pipeline stage.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                EdtChannelsIn(id) {
                    PipelineStage {
                        parent_instance : instance_name ;
                        leaf_instance_name : instance_name ;
                        pipeline_clock : port_pin_name ;
                    }
                }
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT channel in pipeline stage. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Input Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Arguments

- `parent_instance : instance_name ;`

An optional property that specifies the parent instance. This is the name of the instance where the pipeline IP will be generated. If it is not specified, the IP will be placed in the current design scope.

- `leaf_instance_name : instance_name ;`

An optional property that specifies the instance name of the pipeline IP. If not specified, the leaf instance name is auto generated with this format
`<top_module_name>_<dft_spec_id>_tessent_<instrument_name>_<edt_ctrl_id>_<channel_type>_<channel_bit>_pipe_inst_<pipe_index>..`

- `pipeline_clock : port_pin_name ;`

An optional property that specifies the pipeline clock port pin name. This property will override the value of the `pipeline_clock` specified in the `EdtChannelsIn` wrapper. If not specified, it will use the value specified by `EdtChannelsIn` wrapper. If that is not specified, it will default to `edt_clock`. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Input Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Examples

The following example uses the PipelineStage wrapper to define pipeline stages for the EDT input and output channels.

```

DftSpecification(CoreB,rtl) {
    EDT {
        Controller(1) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst1;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[3:0];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe1 ;
                        pipeline_clock : edt_clock;
                    }
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe2 ;
                        pipeline_clock : edt_clock;
                    }
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[3:0];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe3 ;
                        pipeline_clock : edt_clock;
                    }
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe4 ;
                        pipeline_clock : edt_clock;
                    }
                }
            }
        }
        Controller(2) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst2;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[15:12];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe5 ;
                        pipeline_clock : edt_clock;
                    }
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[15:12];
                }
            }
        }
    }
}

```

}

EdtChannelsOut

This wrapper is used to add and specify properties for the EDT channel out ports.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                EdtChannelsOut(id_range) {
                    port_pin_name : port_pin_name ;
                    pipeline_clock : port_pin_name ;
                    PipelineStage {
                    }
                }
            }
        }
    }
}
```

Description

The `process_dft_specification` command for EDT creates bus ports for EDT channel connections on the block level as specified in the configuration data. If the bus ports exist already, they are used for EDT channel connections. If they do not exist, they will be created on the block level by `process_dft_specification`.

You specify the bus names with the `port_pin_name` properties with this configuration data wrapper. The ports must have sufficient bits to make all of the specified EDT channel output connections. EDT `process_dft_specification` cannot enhance existing ports with additional bits.

You can specify the bus ports to be created with any bit range as long as it is a valid Verilog range. They can start with 0 or any other number. Any range is acceptable as long as the EDT channel range width matches the specified connection width. The order you specify the bus ports has an impact on the connection order made to the EDT IP.

The (*id*) component of the `EdtChannelsOut` wrapper accepts a range of positive integer values using a single value or a colon-separated pair of values. For example:

```
EdtChannelsOut(8:7) - two EDT output channels
```

When you specify at least one output connection, you must specify all values and ranges.

Note

 The EDT `process_dft_specification` does not expand existing bus ports. An error will be reported if you use more bits than are on the existing bus.

Arguments

- `port_pin_name` : *port_pin_name* ;
A property that specifies the signals connected to the `edt_channels_out` port of the EDT controller.
- `pipeline_clock` : *port_pin_name* ;
A property that specifies the pipeline clock. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Output Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Examples

Example 1

In this example, the tool connects the first four EDT output channels to the `out*` scalar ports.

```
EdtChannelsOut(1:4) {  
    port_pin_name : out4, out3, out2, out1;  
}
```

The EDT channel index order is reversed. Consequently, the connections are made as follows:

```
edt_ctrl_inst/edt_channels_out[3] -> out1  
edt_ctrl_inst/edt_channels_out[2] -> out2  
edt_ctrl_inst/edt_channels_out[1] -> out3  
edt_ctrl_inst/edt_channels_out[0] -> out4
```

Example 2

This example shows how to create two EDT Controllers with new bus ports.

```
DftSpecification(CoreB,rtl) {
    EDT {
        Controller(1) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst1;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[3:0];
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[3:0];
                }
            }
        }
        Controller(2) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst2;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[15:12];
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[15:12];
                }
            }
        }
    }
}
```

With this configuration data in place, two new EDT controllers will be created when you run the process_dft_specification command, edt_ctrl_inst1 and edt_ctrl_inst2. The output EDT channels in each controller are connected to the created bus ports named “new_output_bus” by the EdtChannelsOut wrapper. This shows the connections made for both controllers:

```
edt_ctrl_inst1/edt_channels_out[3] -> new_output_bus[15]
edt_ctrl_inst1/edt_channels_out[2] -> new_output_bus[14]
edt_ctrl_inst1/edt_channels_out[2] -> new_output_bus[13]
edt_ctrl_inst1/edt_channels_out[0] -> new_output_bus[12]
edt_ctrl_inst2/edt_channels_out[3] -> new_output_bus[15]
edt_ctrl_inst2/edt_channels_out[2] -> new_output_bus[14]
edt_ctrl_inst2/edt_channels_out[1] -> new_output_bus[13]
edt_ctrl_inst2/edt_channels_out[0] -> new_output_bus[12]
```

The new bus ports will have a range of 16 bits (15:0) each because the MSB specified is 15 and the LSB is 0 for both of them. The tool will always create every bus port from its MSB to LSB bits as specified in the config data.

PipelineStage

This wrapper is used to add and specify properties for the EDT channel out pipeline stage.

Usage

```
DftSpecification(module_name, id) {
    EDT {
        Controller(id) {
            Connections {
                EdtChannelsOut(id) {
                    PipelineStage {
                        parent_instance : instance_name ;
                        leaf_instance_name : instance_name ;
                        pipeline_clock : port_pin_name ;
                    }
                }
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the EDT channel out pipeline stage. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Output Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Arguments

- `parent_instance : instance_name ;`
A property that specifies the parent instance. This is the name of the instance where this IP will be generated. If it is not specified, the IP will be placed in the current design scope.
- `leaf_instance_name : instance_name ;`
An optional property that specifies the instance name of the pipeline IP. If not specified, the leaf instance name is auto generated with this format
`<top_module_name>_<dft_spec_id>_tessent_<instrument_name>_<edt_ctrl_id>_<channel_type>_<channel_bit>_pipe_inst_<pipe_index>`.
- `pipeline_clock : port_pin_name ;`
A property that specifies the pipeline clock. This property overrides the value for the `pipeline_clock` specified in the `EdtChannelsOut` wrapper. See “[Pipeline Stage Insertion](#)” and “[Lockup Cells on the Output Side of the EDT Controller](#)” in the *Tessent TestKompress User’s Manual* for more information.

Examples

The following example uses the `PipelineStage` wrapper to define pipeline stages for the EDT input and output channels.

```
DftSpecification(CoreB,rtl) {
    EDT {
        Controller(1) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst1;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[3:0];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe1 ;
                        pipeline_clock : edt_clock;
                    }
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe2 ;
                        pipeline_clock : edt_clock;
                    }
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[3:0];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe3 ;
                        pipeline_clock : edt_clock;
                    }
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe4 ;
                        pipeline_clock : edt_clock;
                    }
                }
            }
        }
        Controller(2) {
            scan_chain_count: 100;
            longest_chain_range: 100,110;
            input_channel_count: 4;
            output_channel_count: 4;
            leaf_instance_name: edt_ctrl_inst2;
            Connections {
                EdtChannelsIn(4:1) {
                    port_pin_name: new_input_bus[15:12];
                    PipelineStage {
                        parent_instance : block1;
                        leaf_instance_name : pipe5 ;
                        pipeline_clock : edt_clock;
                    }
                }
                EdtChannelsOut(4:1) {
                    port_pin_name: new_output_bus[15:12];
                }
            }
        }
    }
}
```

}

ItagNetwork

A wrapper that specifies the IJTAG network to be built and, optionally, inserted into the design.

Usage

```
DftSpecification(module_name,id) {
    ItagNetwork {
        ImplementationOptions {
            scan_path_retiming : latch | flop ; // default: latch
        }
        DataInPorts {
        }
        DataOutPorts {
        }
        HostScanInterface(id) {           // repeatable
            Interface { *DefSpec
            }
            DesignInstance(instance_name) { // repeatable
            }
            ScanMux(id) {               // repeatable
            }
            Sib(id) {                  // repeatable
            }
            Tdr(id) {                  // repeatable
            }
            Tap(id) {                  // repeatable
            }
        }
    }
}
```

Description

A wrapper that specifies the IJTAG network to be built and, optionally, inserted into the design.

This wrapper can be specified once for each DftSpecification wrapper. As illustrated in the syntax above, the ItagNetwork specification consists of one [ItagNetwork/DataInPorts](#) wrapper, one [ItagNetwork/DataOutPorts](#) wrapper, one [ImplementationOptions](#) wrapper, and one or more HostScanInterface wrappers.

A HostScanInterface wrapper defines a scan interface below which a network of IJTAG nodes are serially accessible. The [HostScanInterface/Interface](#) wrapper is used to specify the end points of the HostScanInterface. By default, the wrapper defines a set of ports that may or may not already exist on the current design, but you can also use this wrapper to specify how to connect to an existing host interface present inside the design. For more details and example usage, see the description of the [HostScanInterface/Interface](#) wrapper. If the [HostScanInterface/Interface](#) wrapper only consists of ports on the current design, the [add_icl_scan_interfaces](#) and the [set_icl_scan_interface_ports](#) commands are issued during insertion such that, if you perform ICL extraction once you are done with insertion, the ScanInterface in the extracted ICL module will be named using the id of the HostScanInterface you specified in the DftSpecification.

The other five wrappers, [DesignInstance](#), [ScanMux](#), [Sib](#), [Tap](#), and [Tdr](#), define nodes serially connected below the HostScanInterface. They can be specified in any order, and they are all repeatable; however, their relative order defines the scan path connectivity. The scan-out of the first element declared in the HostScanInterface wrapper sources the scan-out port of the HostScanInterface. The scan-in port of the last element declared in the HostScanInterface wrapper is sourced by the scan-in port of the HostScanInterface.

The SIB, ScanMux, and TAP nodes become host interfaces to the client interfaces of the nodes declared inside them. See the description of the [Sib](#), and [ScanMux](#), [Tap](#) wrappers for more details. The example below illustrates the hierarchical nature of the IJTAG network.

Note

 If you want to insert BoundaryScan and the TAP controller, and connect the TAP to internal pins, follow the instructions in [Example 2](#).

Scan Testability of the IJTAG Network

Elements of the IJTAG network can be used to configure the circuit in its various scan modes. The example shown in [Figure 10-1](#) illustrates the use of a TDR to generate the various scan mode signals. In this example, the portion of the network that gives access to this TDR must remain active during scan test and must remain unaffected by the scan test modes like any other test_setup hardware you may use. If the property `keep_active_during_scan_test` is set on a node, the host access path to the node is traced and every node in the access path is marked with the `keep_active_during_scan_test` attribute. The ICL modules are generated with this attribute and collected into the merged ICL file created by ICL extract. You can later use this information to declare those modules as non-scan during scan insertion, and force them to be kept in the graybox during graybox generation.

When the id of a Tdr wrapper is named “`scan_modes`”, the `keep_active_during_scan_test` property defaults to on for that TDR. This, in turn, forces all nodes needed to access it from the HostScanInterface to have `keep_active_during_scan_test` set to on.

Once you have extracted the final ICL using ICL extract, executing the command “`get_icl_instances -filter keep_active_during_scan_test`” will generate a collection of all ICL instances having `keep_active_during_scan_test` set to on. You can then use the `tessent_design_instance` attribute on the ICL instances to find the corresponding design instances and pass this list to your scan insertion and/or graybox generation step. For a complete list of attributes available on the ICL instance object type, see the [icl_instance](#) data model.

Arguments

None.

Examples

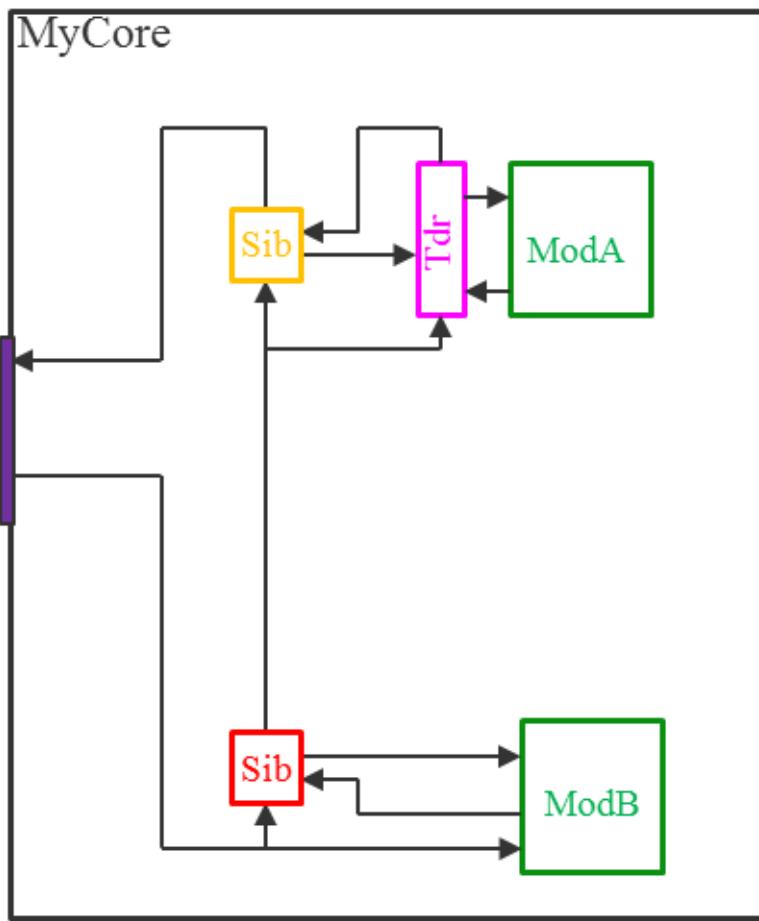
Example 1

The following example shows the DftSpecification of a simple IJTAG network having two SIBs connected in series; the first SIB hosts a design instance, and the second SIB hosts a TDR which in turn sources and observes the data signals of a second design instance. The example assumes the design already includes those two design instances and that an ICL file describing their ports exists for each of them. [Figure 10-1](#) provides the schematic for this example.

The instance “u1/bottom_green” is an instance of module ModB and has a single ScanInterface described in its ICL file; its ScanInterface will be hooked up to the host interface of the red SIB.

The instance “u2/top_green” is an instance of module ModA and has four DataInPorts and two DataOutPorts described in its ICL file. The DataOutPorts of the pink TDR is connected to the DataInPorts of ModA and vice versa. See the description of the [DesignInstance](#) wrapper for information about how the ICL files are automatically searched and matched to the design instance during the [set_current_design](#) invocation.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Sib(orange) {
                Tdr(pink) {
                    DataOutPorts {
                        connection(0) : u2/top_green/en;
                        connection(3:1) : u2/top_green	mode[2:0];
                    }
                    DataInPorts {
                        connection(0) : u2/top_green/go;
                        connection(1) : u2/top_green/done;
                    }
                }
            }
            Sib(red) {
                DesignInstance (u1/bottom_green) {}
            }
        }
    }
}
```

Figure 10-1. IJTAG Network Example Schematic

Example 2

The following example inserts the TAP and the Boundary Scan chain in one pass using a DftSpecification wrapper. This example shows the steps to follow when you want to connect the TAP controller to internal pins which are not directly on the pads associated to the TAP ports. You typically want to do this if you are using a compliance enable pin and the logic to enable the TAP is already present in the design. If the gating logic is not already present, you use the `-active_high_compliance_enables` and/or the `-active_low_compliance_enables` switches of the [create_dft_specification](#) command.

The following example sets the `function` attribute on the top-level ports to identify the top-level TAP port in Step 1. This example connects the TAP to internal pins which are enabled when the top-level port “myCE1” is asserted high. Step 1b shows how your compliance enable port is described such that it gets reported as such in the BSDL file. In this example, it is assumed you already have this logic in the design and that you want to connect the TAP to it. The recommended flow is to not insert this logic manually and let the tool auto create and insert it by using the `-active_high_compliance_enables` and/or the `-active_low_compliance_enables` switches of the [create_dft_specification](#) command. When you do that, the flow is fully automated. When you created your own compliance enable logic as shown in [Figure 10-2](#) on

page 3232, you need to either describe that module in ICL or assert the compliance enable ports before running ICL extraction. The explanation as to when to use which method is described at the bottom of this example

In Step2, you call “set spec [create_dft_specification]” to create the DftSpecification that includes a HostScanInterface wrapper with a TAP wrapper inside it. Step3 below augments the IjtagNetwork wrapper with a second HostScanInterface where the Interface properties are used to specify the connections to internal pins. Step 4 shows the use of the [move_config_element](#) command to move the TAP wrapper that exists in the original HostScanInterface to the new one. Finally, Step 5 reports the final DftSpecification. The HostScanInterface(tap) is used to identify the top-level TAP ports which will be reported in the BSDL file and the second HostScanInterface(tap_internal) specifies where to connect the client scan interface of the Tap controller.

```
# Step 1: set the 'function' attribute on the top level TAP ports
set_attribute_value -name function -value trst jtag_trst
set_attribute_value -name function -value tdi jtag_tdi
set_attribute_value -name function -value tms jtag_tms
set_attribute_value -name function -value tck jtag_tck
set_attribute_value -name function -value tdo jtag_tdo

# Step 1b: declare your compliance enable port so it is documented in
#           the BSDL file
set_boundary_scan_port_options myCE1 -cell_options compliance_enable1

# Step 2: create an initial DFT specification
set_system_mode analysis
set_spec [create_dft_specification]
```

```
# Step 3: add new HostScanInterface with the internal hookup points
read_config_data -in_wrapper ${spec}/IjtagNetwork -from_string {
    HostScanInterface(tap_internal) {
        Interface {
            tck : jtag_and0/o;
            trst : jtag_and3/o;
            tms : jtag_and1/o;
            tdi : jtag_and2/o;
            tdo : jtag_buf5/a;
            tdo_en : jtag_buf4/a;
            tdo_en_polarity : active_high;
        }
    }
}

# step 4: move the Tap to the new HostScanInterface
move_config_element ${spec}/IjtagNetwork/HostScanInterface(tap)/Tap(main) \
    -in_wrapper ${spec}/IjtagNetwork/HostScanInterface(tap_internal)

# step 5: report the final DftSpecification.
report_config_data $spec
The resulting DFT specification will look like:
DftSpecification(top,rtl) {
    IjtagNetwork {
        HostScanInterface(tap) {
            Interface {
                tck : jtag_tck;
                trst : jtag_trst;
                tms : jtag_tms;
                tdi : jtag_tdi;
                tdo : jtag_tdo;
            }
        }
        HostScanInterface(tap_internal) {
            Interface {
                tck : jtag_and0/o;
                trst : jtag_and3/o;
                tms : jtag_and1/o;
                tdi : jtag_and2/o;
                tdo : jtag_buf5/a;
                tdo_en : jtag_buf4/a;
                tdo_en_polarity : active_high;
            }
            Tap(main) {
                HostIjtag(1) {
                }
                HostBscan {
                }
            }
        }
    }
    BoundaryScan {
        ijtag_host_node : Tap(main)/HostBscan;
        pin_order_file : ../data/top.pinorder;
        BoundaryScanCellOptions {
            myCE1 : compliance_enable1;
        }
    }
}
```

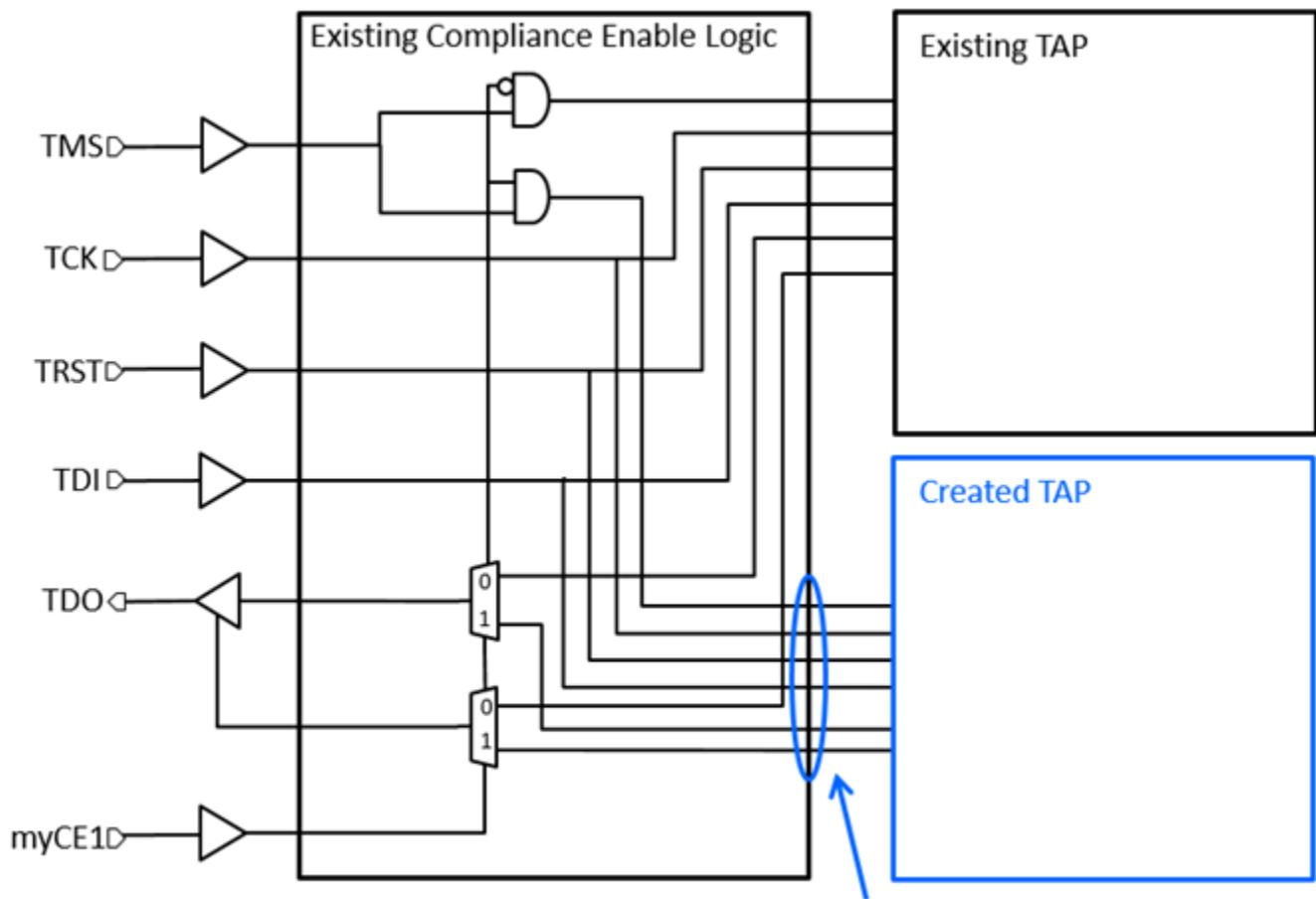
The above example explains how to connect the newly created TAP to internal pins which typically happens when you have compliance enable logic already inserted into the design. This is allowed but you have to deal with this logic before running the [extract_icl](#) command. Realize, however, that the flow is fully automated when you let the tool insert the compliance enable logic using the `-active_high_compliance_enables` and/or the `-active_low_compliance_enables` switches of the [create_dft_specification](#) command.

[Figure 10-2](#) on page 3232 shows the typical logic used to implement compliance enabling when you want to switch between an existing TAP and the Tessent Shell Tap. You may also be using this logic to switch between arbitrary functional logic and the Tessent Shell Tap. If there is no existing TAP or if the existing TAP does not have an ICL view that you want to be represented in the extract ICL file, you simply need to use the [add_input_constraints](#) command to assert the compliance enable ports to their active value before running the [extract_icl](#) command. If the enable is controlled by some internal logic, you need to use a test_setup proc file to describe the vectors needed to activate it. If on the other hand you have an ICL model for the existing TAP and you want it described in the extracted ICL file, you must model the multiplexing module with an ICL file similar the ICL module that follow:

```
Module compliance_enable {
    // from the pads
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo {
        Source ComplianceOutputSelect;
        Attribute forced_high_output_port_list = "tdo_en";
    }
    TMSPort      tms;
    TRSTPort     trst {
        Attribute connection_rule_option = "allowed_tied_high";
    }
    DataInPort   ce;
    ScanInterface tap_pins {
        Port tdi;
        Port tdo;
        Port tms;
        Port tck;
        Port trst;
    }
    // to the existing TAP
    ToTCKPort    tck_other;
    ToTMSPort    tms_other;
    ToTRSTPort   trst_other;
    ScanInPort   tdo_other;
    ScanOutPort  tdi_other {
        Source tdi;
    }
}
```

```
ScanInterface tap_pins_other {
    Port tdi_other;
    Port tdo_other;
    Port tms_other;
    Port tck_other;
    Port trst_other;
}
// to the Tessent Shell generated TAP
ToTCKPort tck_ts;
ToTMSPort tms_ts;
ToTRSTPort trst_ts;
ScanInPort tdo_ts;
ScanOutPort tdi_ts {
    Source tdi;
}
ScanInterface tap_pins_ts {
    Port tdi_ts;
    Port tdo_ts;
    Port tms_ts;
    Port tck_ts;
    Port trst_ts;
}
// the mux for the tdo
ScanMux ComplianceOutputSelect SelectedBy ce {
    1'b1 : tdo_ts;
    1'b0 : tdo_other;
}
}
```

Figure 10-2. Existing Compliance Enable Modules



Pins where the new Tap is specified to be connected

Related Topics

[DesignInstance](#)

[Sib](#)

[ScanMux](#)

[Tdr](#)

IjtagNetwork/ImplementationOptions

A wrapper that specifies implementation options when building the IJTAG network.

Usage

```
IjtagNetwork {
    ImplementationOptions {
        scan_path_retiming : latch | flop ;
    }
}
```

Description

A wrapper that specifies implementation options when building the IJTAG network. The options consist of the use of latches or flip-flops for scan path retiming.

Arguments

- `scan_path_retiming : latch | flop ;`

A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

This property has significant impact on the TAP controller. When the `scan_path_retiming` property is set or left to default to “latch”, a retiming latch is used on the tdo output of the TAP controller to make sure that the `scan_out` data of any scan register is launched from the negative edge of TCK. It is likely that the IJTAG elements feeding the `scan_in` port of its HostIjtag scan interfaces already have a retiming element on their `scan_out` port. The retiming latch inserted in series before reaching TDO has no effect and simply acts as a redundant retiming latch. If you set the property to “flop”, it is no longer possible to systematically place the retiming element on all TDO paths because it would end up in series with the one that is likely already present in IJTAG elements feeding the `scan_in` port of its HostIjtag scan interfaces. Cascading two retiming latches does not affect the resulting timing but cascading two retiming flops will change the timing. When using flops for `scan_path_retiming`, you must ensure that all `scan_out` ports feeding the `scan_in` port of the HostIjtag scan interfaces already have a retiming element. This is always true when using elements created with the `process_dft_specification` command. Only when a third party IJTAG node is directly connected to the `scan_in` port of a HostIjtag scan interface of the TAP, can you fail this requirement.

Note

 Make sure third party nodes include a retiming element on their `scan_out` when you connect them directly to the `scan_in` port of the HostIjtag scan interface’s Tessent Tap, and you have specified the `scan_path_retiming` property to “flop”, otherwise your elements will not be 1149.1 compliant and will likely not be usable within a board environment.

IjtagNetwork/DataInPorts

A wrapper that specifies data-in ports that may or may not exist as input ports on the current design.

Usage

```
IjtagNetwork {
    DataInPorts {
        count           : integer ;          // default: auto
        port_naming     : port_naming, ... ; // default: ijtag_data_in[%d]
        multiplexing    : on | off | auto ;
        connection(range) : pin_name ;      // repeatable
    }
}
```

Description

A wrapper that specifies data-in ports that may or may not exist as input ports on the current design.

These data-in ports can be used to source data-in ports on child design instances, or to source the select port of a [ScanMux](#) as shown in the example for this wrapper.

Arguments

- **count : *integer* ;**

A property to request a specific number of data-in ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d).

- **port_naming : *port_naming*, ... ;**

A property that specifies the naming of the data-in ports. The port_naming value has two distinct formats: a single string using an index symbol “%d”, or a list of port names with explicit sizes.

When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Din%2d will create Din00, Din01, Din02, and so on. Din%d and Din%1d mean the same thing. The default is “ijtag_data_in[%d]”.

When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify “locked”, you will get exactly one data-in port called “locked” as illustrated in “[Example 2](#)” on page 3285 of the [Tdr/DataInPorts](#) section.

- **multiplexing : on | off | auto ;**

A property that specifies if a multiplexer is to be added when making the connection to the specified *pin_name* of the connection properties. When set to auto, a multiplexer is added

unless the pin is unconnected, connected to a constant or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-in port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified *pin_name* of the connection properties. See “[DataOutPort Connection Multiplexing](#)” on page 3290 in the [Tdr/DataOutPorts](#) section for an illustration of this.

- `connection(range) : pin_name ;`

A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the [process_dft_specification -no_insertion](#) option is not used. If you are using the [process_dft_specification -no_insertion](#) switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

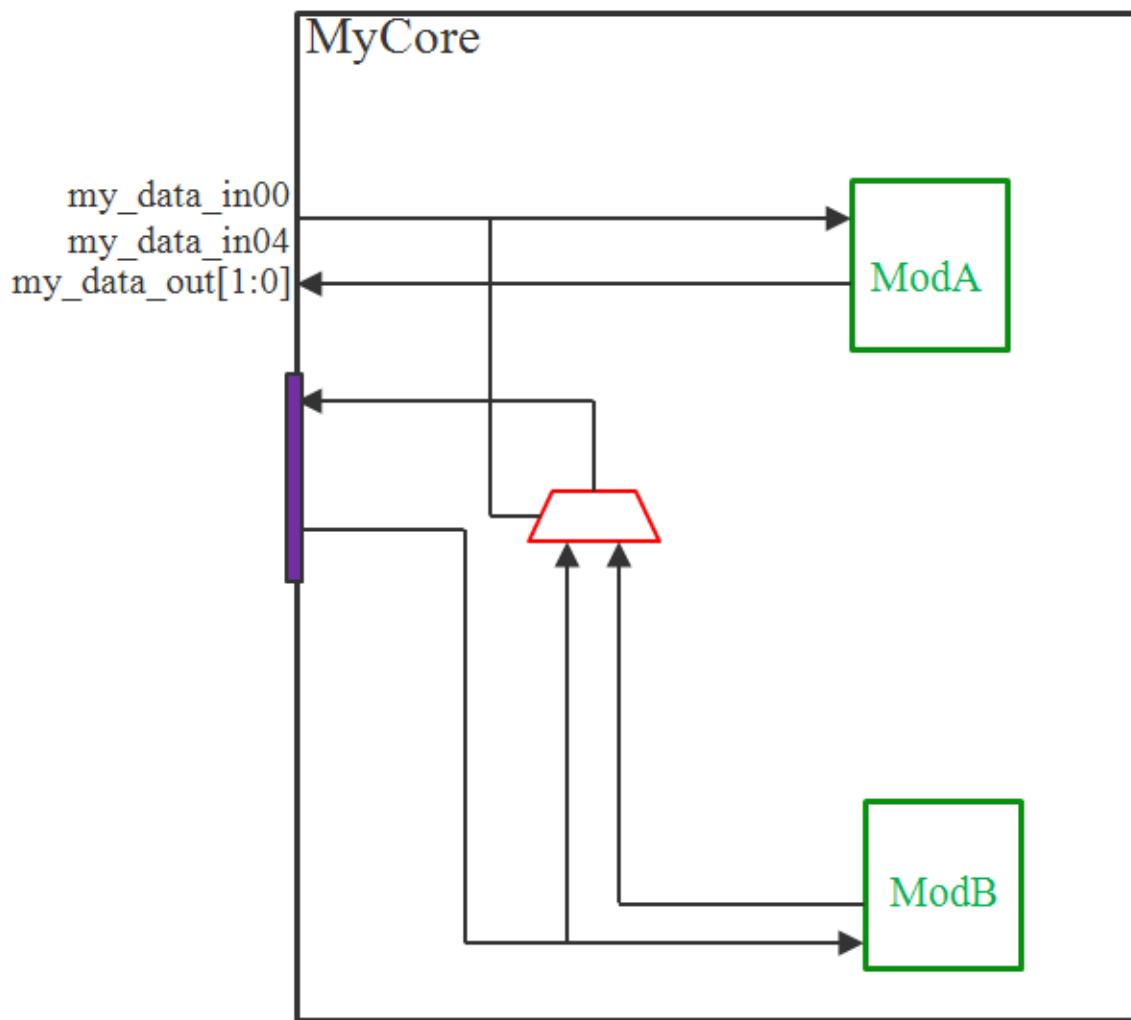
The following example illustrates defining five data-in ports. The ports are named `my_data_in04`, `my_data_in03`, `my_data_in02`, `my_data_in01`, and `my_data_in00`. The port `my_data_in00` is connected to the “en” pin of the `u2/top_green` instance. The ports `my_data_in03`, `my_data_in02`, and `my_data_in01` are connected to the `mode[2:0]` pins of the same instance. If the `count` property was not specified, it would have defaulted to four, corresponding to the largest specified connection range index value of three. In this example, a count of five is specified to request an extra data-in port to be used by the select port of the `ScanMux(red)` used in the `HostScanInterface(purple)`.

Notice that if this core was further integrated into a parent design and a Tdr was used to source the five data-in ports, the property `output_timing(4)` with a value of “`ijtag_scan_selection`” would need to be specified because bit 4 is used to reconfigure the scan network. For further explanation, see “[Update Stage Timing](#)” on page 3277.

The tool is currently unable to determine if the data-in port on a design instance is used to reconfigure the network within the specified design instance but this automation is planned for the future.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        DataInPorts {
            Count           : 5 ;
            port_naming    : my_data_in%2d ;
            connection(0)  : u2/top_green/en ;
            connection(3:1) : u2/top_green/mode[2:0] ;
        }
        DataOutPorts {
            port_naming : my_data_out[%d] ;
            connection(0) : u2/top_green/go ;
            connection(1) : u2/top_green/done ;
        }
        HostScanInterface(purple) {
            ScanMux(red) {
                select : DataIn(4) ;
                Input(1) {
                    DesignInstance (u1/bottom_green) {}
                }
            }
        }
    }
}
```

Figure 10-3. DataInPorts Example Schematic



Related Topics

[IjtagNetwork/DataOutPorts](#)

[Tdr/DataOutPorts](#)

[Tdr/DataInPorts](#)

[Tdr/DecodedSignal](#)

IjtagNetwork/DataOutPorts

A wrapper that specifies data-out ports that may or may not exist as output ports on the current design.

Usage

```
IjtagNetwork {
    DataOutPorts {
        count           : integer ;          // default: auto
        port_naming     : port_naming,...;   // default: ijtag_data_out[%d]
        Connection(range) : pin_name ;      // repeatable
    }
}
```

Description

A wrapper that specifies data-out ports that may or may not exist as output ports on the current design.

These data-out ports can be used to observe data-out ports on child design instances.

Arguments

- **count : *integer*;**

A property to request a specific number of data-out ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the *port_naming* values if they are in a non-indexed format (%d).

- **port_naming : *port_naming*, ... ;**

A property that specifies the naming of the data-out ports. The *port_naming* value as two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

When using the indexed format, you specify *base_name*[%d] when you want to use a bus and you specify *base_name*%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d will create Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

When using a list of port names, you control the exact naming and width of all data-out ports. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in “[DataOutPort Connection Multiplexing](#)” on page 3290 of the [Tdr/DataOutPorts](#) section.

- **connection(*range*) : *pin_name*;**

A repeatable property that specifies the connections to make between the bits of the data-out port and the existing pins inside the design. Because data-out ports are sinks, the same bit

cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the [process_dft_specification -no_insertion](#) option is not used. If you are using the [process_dft_specification -no_insertion](#) switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

The following example illustrates the creation of two data-out ports. The count property is not specified so it defaults to two corresponding to the largest connection range index used. The port naming is specified to define a port named “my_data_out[1:0]”. Each bit is connected to a unique source.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        DataOutPorts {
            port_naming : my_data_out[%d] ;
            connection(0) : u2/top_green/go ;
            connection(1) : u2/top_green/done ;
        }
    }
}
```

Related Topics

[IjtagNetwork/DataInPorts](#)

[Tdr/DataOutPorts](#)

[Tdr/DataInPorts](#)

[Tdr/DecodedSignal](#)

HostScanInterface/Interface

A wrapper used to customize the interface of the HostScanInterface.

Usage

```
IJtagNetwork {
    HostScanInterface(id) {
        Interface {
            design_instance : instance_name ; // default: null
            scan_interface : id ; // default: null
            tck : port_pin_name ; // default: ijtag_tck
            reset : port_pin_name ; // default: ijtag_reset *DefSpec
            select : port_pin_name ; // default: ijtag_sel *DefSpec
            shift_en : port_pin_name ; // default: ijtag_se *DefSpec
            capture_en : port_pin_name ; // default: ijtag_ce *DefSpec
            update_en : port_pin_name ; // default: ijtag_ue *DefSpec
            scan_in : port_pin_name ; // default: ijtag_si *DefSpec
            scan_out : port_pin_name ; // default: ijtag_so *DefSpec
            reset_polarity : active_high | active_low | auto ;
            trst : port_pin_name ; // default: trst *DefSpec
            tms : port_pin_name ; // default: tms *DefSpec
            tdi : port_pin_name ; // default: tdi *DefSpec
            tdo : port_pin_name ; // default: tdo *DefSpec
            tdo_en : port_or_pin_name ; // default: tdo_en
            tdo_en_polarity : active_high | active_low | auto ;
            active_high_compliance_enables : port_pin_name, ... ;
            active_low_compliance_enables : port_pin_name, ... ;
            daisy_chain_with_existing_client : on | off | auto ;
        }
    }
}
```

Description

A wrapper used to customize the interface of the HostScanInterface.

By default, the HostScanInterface connects directly to ports on the current design and follows the default naming shown in the syntax summary above. Those ports may or may not already exist in the current design. If they are not already present, they are created during insertion if the design level is set to "physical_block" or "sub_block". If the design level is set to "chip", ports need to already exist and be connected to pad cells. For more information, see the [set_design_level](#) and [read_cell_library](#) commands.

You can also specify internal pins to connect the HostScanInterface to an existing host interface already present in the design. The figure shows how the Interface wrapper is used to create top-level ports, and to hook up to an existing host port. [Example 2](#) shows how to hook up a SIB to a TAP inserted by ETAssemble in the LV Flow.

To insert a TAP controller on internal pins and perform boundary scan insertion, refer to [Example 2](#) of the [IJtagNetwork](#) wrapper section which shows the required specification manipulations.

When connecting to an internal pin, specifying the `reset_polarity` of the reset pin is required; otherwise, you will get an ICL extract violation later on if the assumed value is not consistent with the actual reset polarity. If specified as `auto`, `reset_polarity` is determined by counting the number of children with an active high and an active low reset polarity and taking the polarity of the larger count.

Parameters

- `design_instance : instance_name ;`

This property defines the HostScanInterface on a ScanInterface defined in an ICL module associated with an instance in the design. The specified `instance_name` must be the name of an instance that already exists in the design, and it must be the instantiation of a module having a corresponding ICL module.

ICL modules are automatically searched and matched to the design modules when the `set_current_design` command is executed. The search paths for the ICL files are specified using the `set_design_sources`-format `icl` command. The module name matching rules are specified using the `set_module_matching_options` command.

A `design_instance` cannot be specified when `process_dft_specification` is run using the `-no_insertion` option.

- `scan_interface : id ;`

This property specifies which ScanInterface to define the HostScanInterface on when the `design_instance` property points to an instance associated with an ICL module having more than one ScanInterface. If the associated ICL module only has one ScanInterface, the `scan_interface` property is not needed.

If the referenced ScanInterface is a client ScanInterface, the host ScanInterface is defined in such a way that the inserted IJTAG nodes will end up behind and in series with the referenced client ScanInterface. If the referenced ScanInterface is a host ScanInterface, the inserted IJTAG nodes will connect to it. This varying behavior based on the fact that the ScanInterface is of type host or client is illustrated in Example 1.

- `tck : port_pin_name ;`

This property defines the port or the pin name to source the IJTAG TCKPort function. The default is “`ijtag_tck`”. See `create_dft_specification` description to learn how it can be automatically set.

- `reset : port_pin_name ;`

This property defines the port or the pin name to source the IJTAG ResetPort function. The default is “`ijtag_reset`”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `select : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG SelectPort function. The default is “`ijtag_sel`”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `shift_en : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG ShiftEnPort function. The default is “ijtag_se”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `capture_en : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG CaptureEnPort function. The default is “ijtag_ce”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `update_en : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG UpdateEnPort function. The default is “ijtag_ue”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `scan_in : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “ijtag_si”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `scan_out : port_pin_name ;`

This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “ijtag_so”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `reset_polarity : active_high | active_low | auto ;`

This property defines the polarity of the ResetPort. When set to auto, the reset_polarity is determined by counting the number of children with an active high or an active low reset port and taking the polarity of the larger count. When the reset property points to an existing reset pin, it is required to specify the reset polarity consistent with the source being used. Failing to specify the proper polarity results in a subsequent ICL extraction DRC violation. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- `trst : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG TRSTPort function. The default is “trst”. This property is only used when the HostScanInterface is for a TAP scan interface. See [create_dft_specification](#) description to learn how it can be automatically set.

When performing boundary scan insertion, the specified port needs to be a top level port. A 5-pin TAP interface will be created for this specification. An internal pin can also be specified, which needs to be a power-on reset pin. A 4-pin TAP interface will be created for this specification.

Note

 When performing boundary scan insertion, if you need to specify an internal pin for trst that connects to a top level port, two insertion passes are needed. The TAP needs to be inserted in the first pass and the boundary scan insertion is performed in a subsequent pass.

- `tms: port_pin_name ;`

This property defines the port or the pin name to receive the IJTAG TMSPort function. The default is “tms”. This property is only used when the HostScanInterface is for a TAP scan interface. See [create_dft_specification](#) description to learn how it can be automatically set.

- `tdi : port_pin_name ;`

This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “tdi”. This property is only used when the HostScanInterface is for a TAP scan interface. See [create_dft_specification](#) description to learn how it can be automatically set.

- `tdo : port_pin_name ;`

This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “tdo”. This property is only used when the HostScanInterface is for a TAP scan interface. See [create_dft_specification](#) description to learn how it can be automatically set.

- `tdo_en : port_or_pin_name ;`

This property defines the port or the pin name to receive the tdo_en function. The default is “tdo_en”. This property is only used when the HostScanInterface is for a TAP scan interface and the tdo port is not driven by an output pad buffer in which case no tdo_en port is created as the tdo_en signal is connected to the enable pin of the output pad buffer.

- `tdo_en_polarity : active_high | active_low | auto ;`

This property defines the polarity of the port or the pin name that receive the tdo_en function. This property is only used when the HostScanInterface is for a TAP scan interface. If the tdo port is driven by an output pad buffer, the tdo_en_polarity is inferred from the polarity of the enable on the TAP. An error is generated if the tdo_en_polarity property is specified to active_high and the enable pin if the output buffer is active_low and vice versa. No error is ever generated if specified to auto. If the tdo port is not driven by an output pad buffer, a tdo_en port is created with the specified active polarity. The polarity active_low is used if the property is specified or defaults to auto.

- `active_high_compliance_enables : port_pin_name, ... ;`

This property defines active high ports or pins that need to be enabled to select the scan path of the HostScanInterface. Specifying it triggers the creation of the Tessent compliance enable module. Use this property only when the HostScanInterface is for a TAP scan interface and is not defined using the design_instance and scan_interface properties.

The Tessent compliance enable module decodes all compliance enable signals, muxes the TDO signal and gates the TMS signal going to the nodes in this HostScanInterface. You would typically use the `active_high_compliance_enables : port_pin_name, ...;` property if you currently have a TAP controller connected to the control ports and wish to use a Tessent TAP for boundary scan tests. Contrary to using a ScanMux for this purpose, the existing TAP does not need to be described in ICL. The expectation of Tessent Shell and the IEEE 1687 standard is that all other TAPs sharing the control ports have their TMS signal gated low when all compliance enable signals are active. If that is not the case, it is suggested that you use a ScanMux configuration instead.

- `active_low_compliance_enables : port_pin_name, ... ;`

This property defines active low ports or pins that need to be enabled to select the scan path of this HostScanInterface. See the `active_high_compliance_enables` property for more details. This property is only used when the HostScanInterface is for a TAP scan interface and is not defined using the `design_instance` and `scan_interface` properties.

- `daisy_chain_with_existing_client : on | off | auto ;`

A string property affecting how the connection to the HostScanInterface is done. When set to off, the source of the `scan_out` pin is first disconnected before hooking up the scan-out port of the network to it.

When set to on, it is assumed a valid scan-out source is already connected to the specified scan-out port. The source of the scan-out is used as the scan-in of the newly inserted IJTAG network.

When set or left to default to auto, the scan-out port of the HostScanInterface is checked and if it seems to already have a source (its `has_functional_source` attribute is true), the On value is assumed otherwise the Off value is assumed.

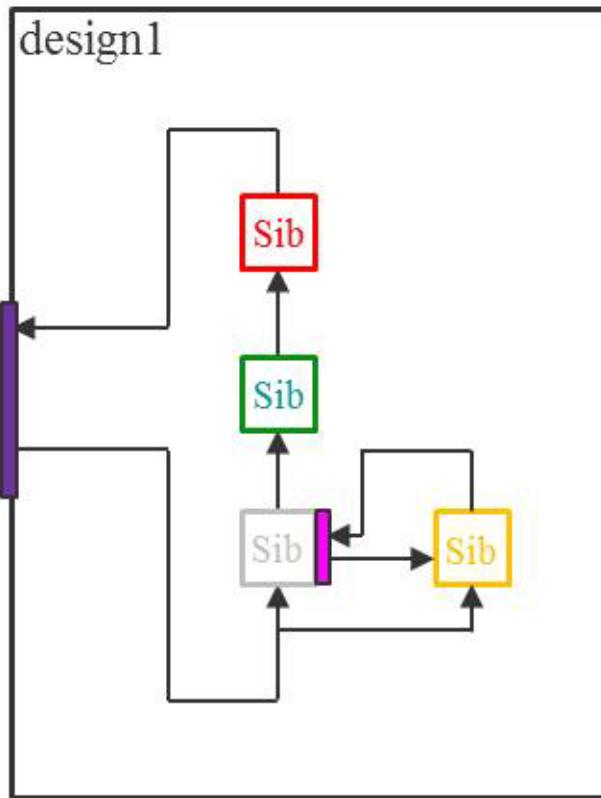
Examples

Example 1

The following example has three HostScanInterface wrappers. The first wrapper called purple is used to insert a SIB called red in series with whatever is already connected to the purple HostScanInterface (the gray SIB in [Figure 10-4](#)). The second wrapper, called pink, is used to connect a SIB called orange to the Host ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass. The third wrapper, called gray, is used to connect a SIB called green in series with the Client ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass.

```
DftSpecification(design1,pass2) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Interface {
                scan_in      : si ;
                scan_out     : so ;
                capture_en   : ce ;
                shift_en     : se ;
                update_en    : ue ;
                reset        : rstn ;
                select       : sel ;
                daisy_chain_with_existing_client : on ;
            }
            Sib(red) {
            }
        }
        HostScanInterface(pink) {
            Interface {
                scan_in      : si ;
                scan_out     : tesson_pass1_sib_first_inst/ijtag_from_so ;
                capture_en   : ce ;
                shift_en     : se ;
                update_en    : ue ;
                reset        : rstn ;
                select       : tesson_pass1_sib_first_inst/ijtag_to_sel ;
            }
            Sib(orange) {
            }
        }
        HostScanInterface (gray){
            Interface {
                design_instance : tesson_pass1_sib_first_inst ;
                scan_interface  : client ;
            }
            Sib(green) {
            }
        }
    }
}
```

Figure 10-4. HostScanInterface/Interface Example 1 Schematic



Example 2

This example shows how to connect a SIB to the TAP controller inserted by ETAssemble as part of one of the LV flows. The first DftSpecification wrapper uses the `design_instance` syntax. The second DftSpecification wrapper uses explicit function connections but results in the same implementation as the first one. Although the second method is more verbose, it works even when the connections are not all made to a single instance having an associated ICL module, which is a requirement for the first method.

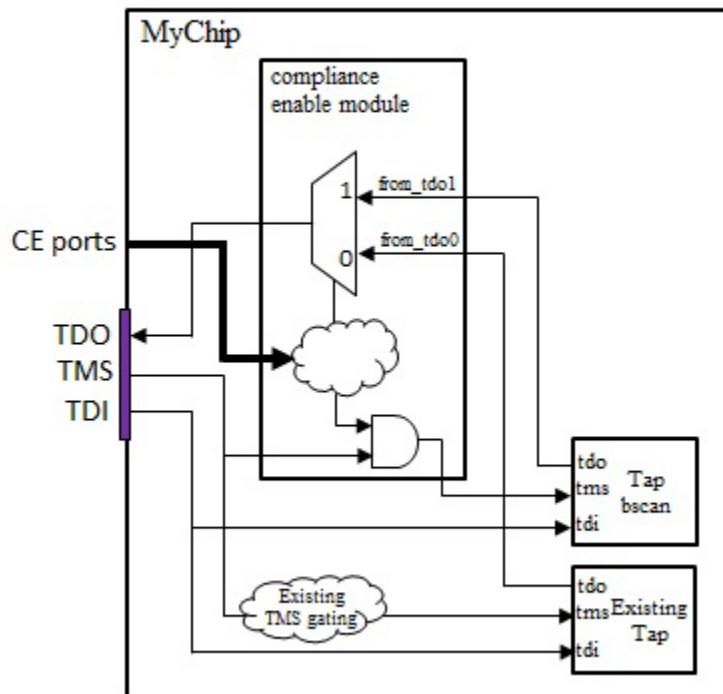
```
DftSpecification(my_design,post_etassemble) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Interface {
                design_instance : LVISION_JTAP_INST ;
                scan_interface : Bist4 ;
            }
            Sib(1) {
            }
        }
    }
}
DftSpecification(my_design,post_etassemble) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Interface {
                scan_in      : LVISION_JTAP_INST/tdi ;
                scan_out     : LVISION_JTAP_INST/fromBist4 ;
                select       : LVISION_JTAP_INST/bistEn4 ;
                capture_en   : LVISION_JTAP_INST/captureDR2Edge ;
                shift_en     : LVISION_JTAP_INST/shiftDR2Edge ;
                update_en    : LVISION_JTAP_INST/updateDREnable ;
                reset        : LVISION_JTAP_INST/testLogicResetInv ;
                reset_polarity : active_low ;
                tck          : LVISION_JTAP_INST/tck ;
            }
            Sib(1) {
            }
        }
    }
}
```

Example 3

This example shows how to insert a TAP with boundary scan that shares ports with an existing TAP controller through compliance enable ports. Notice that the existing TAP has TMS gating low when the compliance enable values are not selecting it. If you are inserting boundary scan, the compliance enable ports are required to be documented as shown in the BoundaryScanCellOptions.

```
DftSpecification(MyChip,bscan_tap) {
    IjtagNetwork {
        HostScanInterface(tap) {
            Interface {
                tck : TCK ;
                trst : TRST ;
                tms : TMS ;
                tdi : TDI ;
                tdo : TDO ;
                active_high_compliance_enables : CE0, CE1 ;
                active_low_compliance_enables : CE2, CE3 ;
            }
            Tap(bscan) {
                HostIjtag(1) {
                }
                HostBscan {
                }
            }
        }
    }
    BoundaryScan {
        ijtag_host_node : Tap(bscan)/HostBscan;
        BoundaryScanCellOptions {
            CE0 : compliance_enable1 ;
            CE1 : compliance_enable1 ;
            CE2 : compliance_enable0 ;
            CE3 : compliance_enable0 ;
        }
    }
}
```

Figure 10-5. HostScanInterface/Interface Example 3 Schematic



Related Topics

[ScanMux/Interface](#)

[Tdr/Interface](#)

[Sib/Interface](#)

DesignInstance

A wrapper that specifies an instance in the design that has a ScanInterface to connect to.

Usage

```
IjtagNetwork {
    HostScanInterface(id) {
        DesignInstance(instance_name) {
            scan_interface : scan_interface_name ;
        }
    }
}
```

Description

A wrapper that specifies an instance in the design that has a ScanInterface to connect to.

The *instance_name* string is a design instance name relative to the current design. When the [process_dft_specification](#) command is executed without the *-no_insertion* option, the tool verifies that the specified *instance_name* exists within the design. The tool also verifies that the associated design module has a matched ICL module with at least one ScanInterface in it. If there is only one scan interface in the ICL module, you do not need to specify the *scan_interface* property, but if there is more than one ScanInterface, you must use the *scan_interface* property to identify which one to connect to.

ICL modules are automatically searched and matched to the design modules when the [set_current_design](#) command is executed. The search paths for the ICL files are specified using the [set_design_sources](#) -format *icl* command. The module name matching rules are specified using the [set_module_matching_options](#) command.

Arguments

- *instance_name*
A string that refers to a design instance relative to the current design.
- *scan_interface : scan_interface_name ;*
A property that is used only when the referenced design instance is an instance of a module having more than one ScanInterface declared in its ICL module. You must repeat the DesignInstance wrapper for each *scan_interface_name* found in the ICL module, if you want to connect all of them to the network.

If you have a design instance with multiple scan interfaces and they share a common scan-in port, you cannot connect them to a SIB because this would required independent scan-in ports. Instead, you must connect them all to inputs of a scan mux, so that you end up with one scan-in port and one scan-out port. This is illustrated in the example that follows.

Examples

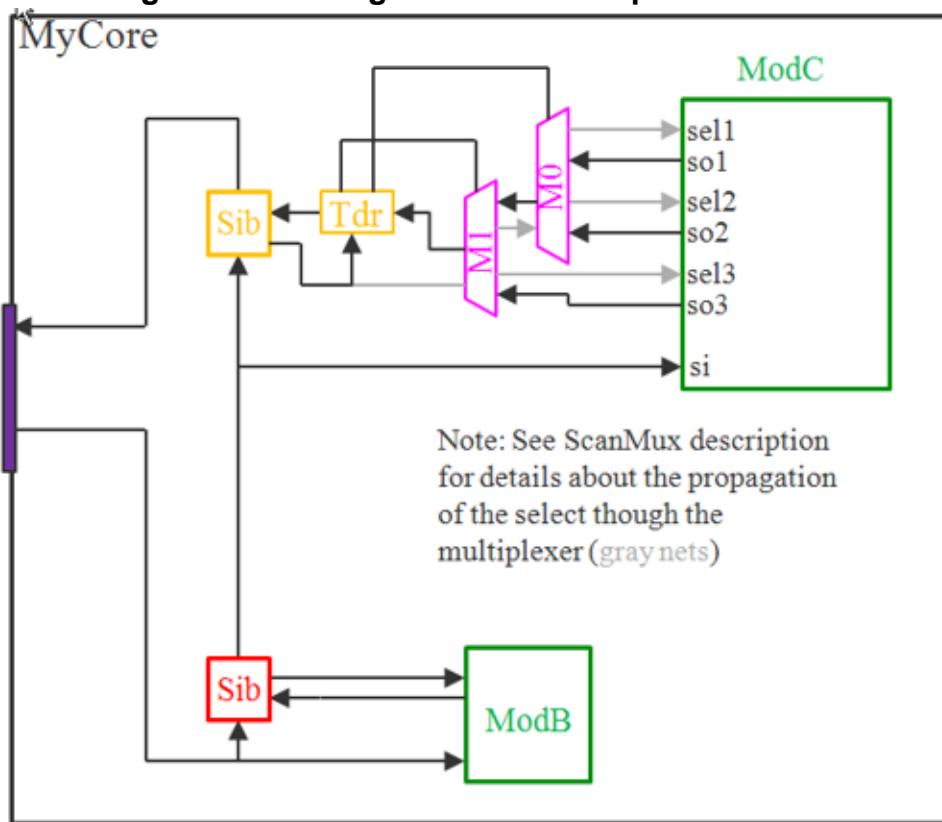
The following example illustrates connecting the ScanInterface of a DesignInstance to a SIB. It also shows how a design instance that has three scan interfaces sharing a common scan-in port is first connected to two scan multiplexers before being connected to a SIB.

The select of the multiplexers are sourced by the DataOutPorts of the orange TDR which automatically sets its output_timing setting to “ijtag_scan_selection”. For a description and impact of this setting, see the [Tdr](#) wrapper description and specifically “[Update Stage Timing](#)” on page 3277.

This example also shows the use of ScanMux in the network. It provides a hint about the backward propagation of the select signal though the selected path of the multiplexer. For a complete description of the ScanMux and the backward propagation of the select signals, refer to the [ScanMux](#) wrapper description section.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Sib(orange) {
                Tdr(orange) {
                    DataOutPorts {
                        Count : 2 ;
                    }
                }
            ScanMux(M1) {
                Select : tdr(orange)/DataOut(1) ;
                Input(0) {
                    ScanMux(M0) {
                        Select : tdr(orange)/DataOut(0) ;
                        Input(0) {
                            DesignInstance (u3/top_green) {
                                scan_interface : I1 ;
                            }
                        }
                        Input(1) {
                            DesignInstance (u3/top_green) {
                                scan_interface : I2 ;
                            }
                        }
                    }
                }
                Input(1) {
                    DesignInstance (u3/top_green) {
                        scan_interface : I3 ;
                    }
                }
            }
        }
        Sib(red) {
            DesignInstance (u1/bottom_green) {}
        }
    }
}
```

Figure 10-6. DesignInstance Example Schematic



Related Topics

[HostScanInterface/Interface](#)

[Sib](#)

[ScanMux](#)

[Tdr](#)

ScanMux

A wrapper that specifies the creation and, optionally, the insertion of a scan multiplexer into the design.

Usage

```
IjtagNetwork {
    HostScanInterface(id) {
        ScanMux(id) {
            Interface {
            }
            parent_instance      : instance_name ;
            leaf_instance_name  : leaf_instance_name ;
            select              : source ;
            Attributes {
                attribute_name   : attribute_value ;
            }
            Input (integer) {                                // legal: 0 1
                keep_active_during_scan_test : boolean ; // default: auto
                Sib(id) {
                }
                Tap(id) {
                }
                Tdr(id) {
                }
                ScanMux(id) {
                }
                DesignInstance(instance_name) {
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a scan multiplexer into the design. The Input(0) and Input(1) wrappers are used to defined the nodes that feed into that input of the multiplexer. If you omit an Input wrapper, the multiplexer simply bypasses the elements connected to the other input when the unspecified input is selected. See the example that follows for an illustration of this feature.

The actual schematic of the multiplexer is shown in [Figure 10-7](#) and [Figure 10-8](#). As you can see, the scan multiplexer is actually more than a simple multiplexer. When used in a DR scan path, it also has logic to supply a select line (output_en1 and output_en0) for each set of clients connected to the two scan inputs (input1 and input0). When used in a TAP scan path, it also has logic to supply a to_tms (to_tms1 and to_tms0) for each set of tap clients connected to the two scan inputs (input1 and input0). The output port is the multiplexed scan-out data of the clients to source the scan-in of the host or of the next node in series with the ScanMux.

The select input is sourced by a data signal which is either one of the [IjtagNetwork/DataInPorts](#) of the [IjtagNetwork](#), one of the [IjtagNetwork/DataOutPorts](#) of a [Tdr](#) or a [Tap](#), or an output port

on an existing design instance. The select input is used to control the multiplexer and to create two gated versions of the enable_in or tms inputs. The gated versions are used as the select or tms source of the clients connected to each scan input. The enable_in port is sourced by the host of the ScanMux. If the host is a **Sib**, the enable_in is sourced by the to_select port. If the host is another **ScanMux**, the enable_in port is sourced by one of the “enable_out#” ports. If the host is the **HostScanInterface/Interface**, the enable_in is sourced by the select port. The tms port is sourced by the host of the ScanMux. If the host is a ScanMux, the tms port is sourced by one of the “to_tms#” ports. If the host is the HostScanInterface, the tms is sourced by the tms port.

When the reuse_modules_when_possible property is set or inferred to be on in the **DftSpecification** wrapper, the ScanMux module names are hardcoded to *design_name_design_id_tessent_scanmux_#* where # is an integer incrementing from 1. A unique module is created for each combination of port names in the interface wrapper, attributes in the Attributes wrapper, and the value of the keep_active_during_scan_test property.

When the reuse_modules_when_possible property is set or inferred to be off in the **DftSpecification** wrapper, the ScanMux module names are hardcoded to *design_name_design_id_tessent_scanmux_id*.

Figure 10-7. ScanMux Internal Schematic When Used on a DR Scan Path

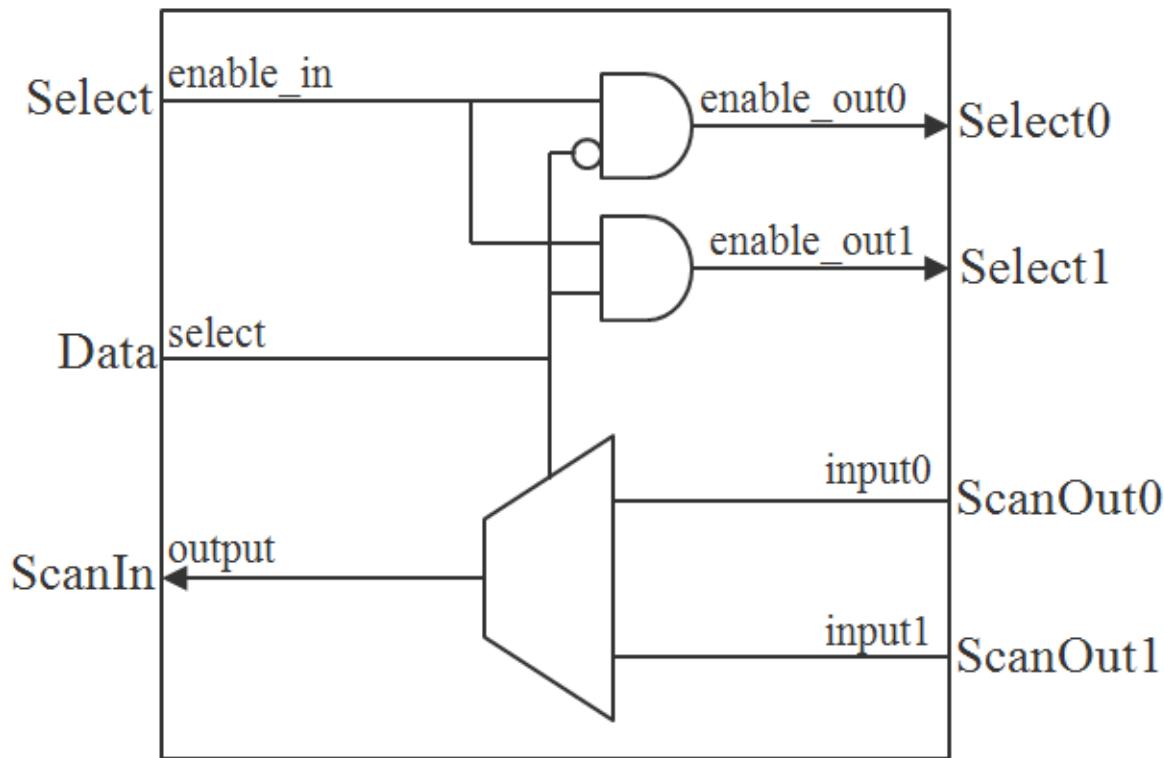
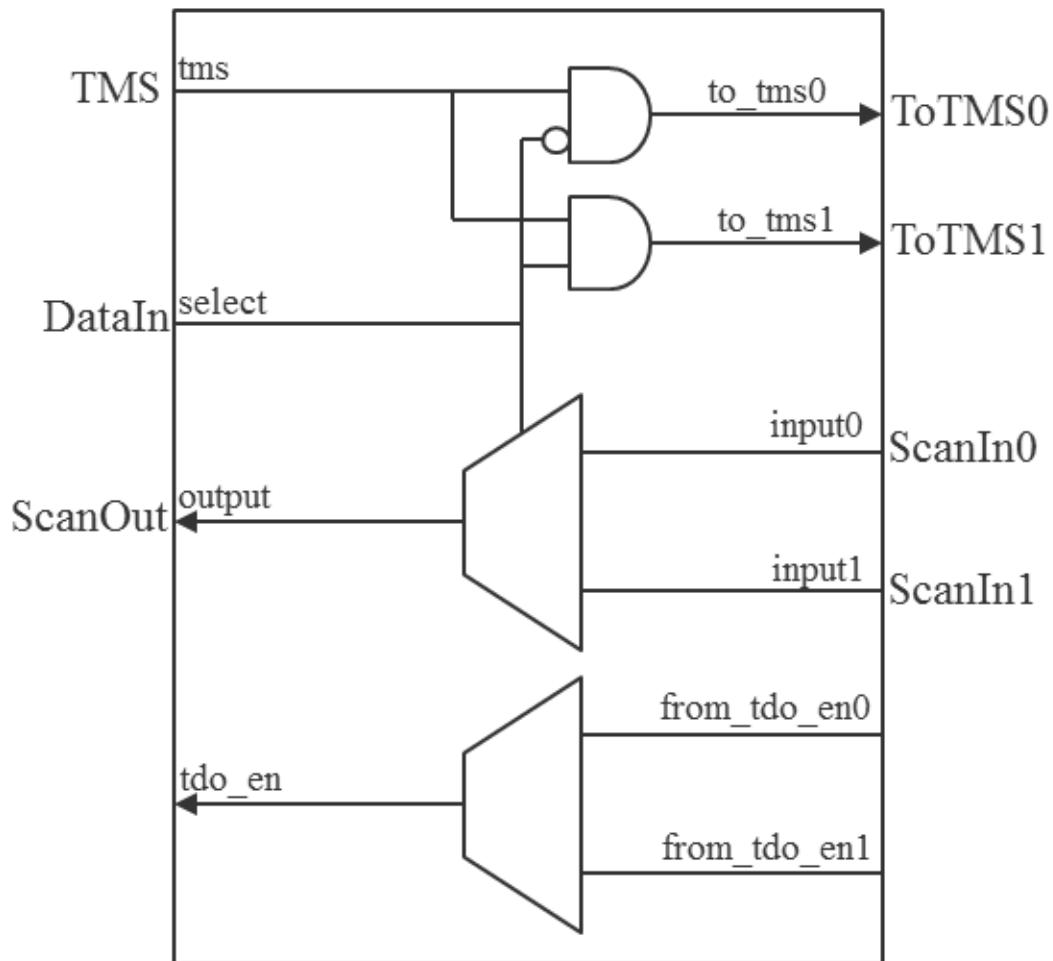


Figure 10-8. ScanMux Internal Schematic When Used on a TAP Scan Path



Arguments

- *id*
A unique id that contains only letters, numbers, and underscores. Every ScanMux wrapper must have a different id inside a common [IjtagNetwork](#) wrapper.
- *parent_instance : instance_name ;*
A string property that specifies the parent instance in which to instantiate the scan mux module. If this property is not unspecified, the scan mux is instantiated in the root module. You can use this property to push a scan multiplexer down near the client interfaces it interacts with.

If the specified *instance_name* is an instance of a repeated module, a warning is displayed during validation and a *uniquify_instances* command is automatically issued as part of the insertion process to uniquify the *instance_name*.

To perform IJTAG network insertion inside a module that is used more than once without unquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the `set_current_design` command in setup mode.

- `leaf_instance_name : leaf_instance_name ;`

A string property that specifies the leaf instance name of the scan multiplexer. The property defaults to `design_name_dft_spec_id_tessent_scanmux_scanmux_id_inst` when unspecified. The `dft_spec_id` string is the id specified in the [DftSpecification](#) wrapper, and the `scanmux_id` is the id of the ScanMux wrapper. Tessent Shell checks for conflicts before instantiating the scan multiplexer and, if needed, applies the unification suffix specified by the [set_insertion_options](#) command.

- `select : source ;`

A string property that specifies the source of the select. The source must match one of the formats described in this table.

Format	Description
<code>DataIn(integer)</code>	Specifies that the source of the select is one of the IjtagNetwork/DataInPorts of the IjtagNetwork wrapper. The integer value must be between count -1 and 0 where count is a property that specifies the number of DataInPorts.
<code>Tdr(id)/DataOut(integer)</code>	Specifies that the source of the select is one of the IjtagNetwork/DataOutPorts of a Tdr . The integer value must be between count -1 and 0 where count is a property that specifies the number of DataOutPorts on the <code>Tdr(id)</code> .
<code>Tdr(id)/DecodeSignal(name)</code>	Specifies that the source of the select is one of the Tdr/DecodedSignal of a Tdr . The name value must be the name of a decoded signal on the <code>Tdr(id)</code> .
<code>Tap(id)/DataOut(integer)</code>	Specifies that the source of the select is one of the IjtagNetwork/DataOutPorts of a Tap . The integer value must be between count -1 and 0 where count is a property that specifies the number of DataOutPorts on the <code>Tap(id)</code> .
<code>DesignInstance(inst_name)/port_name</code>	Specifies that the source of the select is a port on the module with the instance name <code>inst_name</code> . During ICL extraction, this pin must be or must trace to a DataOutPort or a ToSelect port of an ICL module.

- Attributes/attribute_name : *attribute_value* ;

A data wrapper that defines arbitrary attributes in the ScanMux ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using [get_attribute_value_list](#), you must have registered the attribute using the “[register_attribute -object_type icl_module](#)” command. For example, assuming you have registered the attribute att1 on the [icl_module](#) object type, and you have added the attribute att1 with a value of “v1” in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the “[get_icl_modules -filter att1==v1](#)” command.

- Input(0|1)/keep_active_during_scan_test : on | off | [auto](#) ;

A property that specifies that all clients connected to the ScanMux input are to be kept active during scan test modes. You should rarely need to use this property because when it is auto (default), it is automatically set to on if a Tdr with keep_active_during_scan_test set to on traces to it. See section “[Scan Testability of the IJTAG Network](#)” on page 3225 for information about when to use this property.

Examples

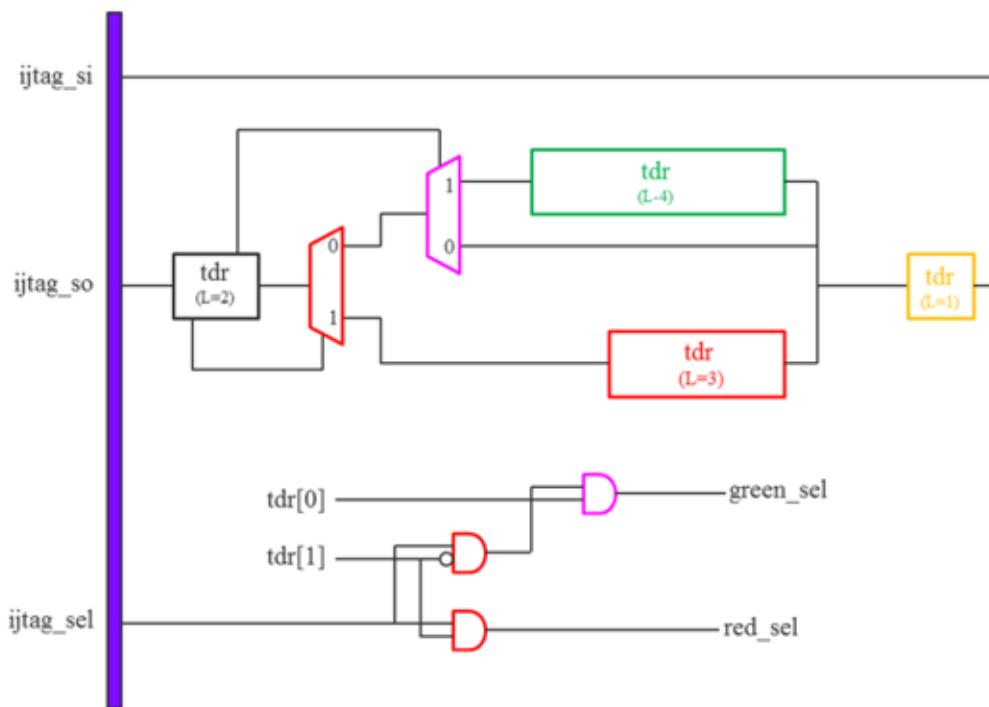
The following example illustrates the use of scan multiplexers in the DR scan paths of the IJTAG network. See [Figure 10-20](#) for an example use in the TAP scan path. The red mux has client nodes specified in both of its Input wrappers which make it a selection multiplexer. The pink mux only has client nodes in its “1” Input wrapper making it a bypass multiplexer. The clients connected to the “1” input are bypassed when the select is low. For both multiplexers, a data-out bit on the black TDR is used as the source of the select signal.

The AND gates shown at the bottom of [Figure 10-9](#) illustrate the distributed gating mechanism used to generate the select signals for the multiplexed TDRs. These AND gates correspond to the AND gates shown in [Figure 10-7](#).

```

DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Tdr(black) {
                DataOutPorts {
                    count : 2 ;
                }
            }
            ScanMux(red) {
                select : tdr(black)/DataOut(1) ;
                Input(1) {
                    Tdr(red) {
                        length : 3 ;
                    }
                }
                Input(0) {
                    ScanMux(pink) {
                        select : tdr(black)/DataOut(0) ;
                        Input(1) {
                            Tdr(green) {
                                length : 4 ;
                            }
                        }
                    }
                }
            }
            Tdr(orange) {
            }
        }
    }
}
    
```

Figure 10-9. ScanMux Example Schematic



Related Topics

[DesignInstance](#)

[Sib](#)

[Tdr](#)

[HostScanInterface/Interface](#)

ScanMux/Interface

A wrapper that specifies the name of the ports when creating the ScanMux module.

Usage

```
IjtagNetwork {
    HostScanInterface(id) {
        ScanMux(id) {
            Interface {
                select      : port_name ; // default: mux_select
                input0     : port_name ; // default: mux_in0
                input1     : port_name ; // default: mux_in1
                output      : port_name ; // default: mux_out
                enable_in   : port_name ; // default: enable_in
                enable_out0 : port_name ; // default: enable_out0
                enable_out1 : port_name ; // default: enable_out1
                tms         : port_name ; // default: tms
                to_tms0     : port_name ; // default: to_tms0
                to_tms1     : port_name ; // default: to_tms1
                tdo_en      : port_name ; // default: tdo_en
                from_tdo_en0: port_name ; // default: from_tdo_en0
                from_tdo_en1: port_name ; // default: from_tdo_en1
            }
        }
    }
}
```

Description

A wrapper that specifies the name of the ports when creating the ScanMux module.

All port names can be specified by the user; however, they can only be simple scalar port names.

Arguments

- `select : port_name ;`

This property defines the name of the port labelled “select” in [Figure 10-7](#) and [Figure 10-8](#).
The default is “mux_select”.

- `input0 : port_name ;`

This property defines the name of the port labelled “input0” in [Figure 10-7](#) and [Figure 10-8](#).
The default is “mux_in0”.

- `input1 : port_name ;`

This property defines the name of the port labelled “input1” in [Figure 10-7](#) and [Figure 10-8](#).
The default is “mux_in1”.

- `output : port_name ;`

This property defines the name of the port labelled “output” in [Figure 10-7](#) and [Figure 10-8](#).
The default is “mux_out”.

- `enable_in : port_name ;`
This property defines the name of the port labelled “enable_in” in [Figure 10-7](#). The default is “enable_in”.
- `enable_out0 : port_name ;`
This property defines the name of the port labelled “enable_out0” in [Figure 10-7](#) and [Figure 10-8](#). The default is “enable_out0”.
- `enable_out1 : port_name ;`
This property defines the name of the port labelled “enable_out1” in [Figure 10-7](#). The default is “enable_out1”.
- `tms : port_name ;`
This property defines the name of the port labelled “tms” in [Figure 10-8](#). The default is “tms”.
- `to_tms0 : port_name ;`
This property defines the name of the port labelled “to_tms0” in [Figure 10-8](#). The default is “to_tms0”.
- `to_tms1 : port_name ;`
This property defines the name of the port labelled “to_tms1” in [Figure 10-8](#). The default is “to_tms1”.
- `tdo_en : port_name ;`
This property defines the name of the port labelled “tdo_en” in [Figure 10-8](#). The default is “tdo_en”.
- `from_tdo_en0 : port_name ;`
This property defines the name of the port labelled “from_tdo_en0” in [Figure 10-8](#). The default is “from_tdo_en0”.
- `from_tdo_en1 : port_name ;`
This property defines the name of the port labelled “from_tdo_en1” in [Figure 10-8](#). The default is “from_tdo_en1”.

Examples

The following example illustrates the use of this wrapper to specify custom port names for the scan multiplexer used in a DR Scan Path.

```
IjtagNetwork {
    HostScanInterface(id) {
        ScanMux(id) {
            Interface {
                select      : sm_select ;
                input0     : sm_si0 ;
                input1     : sm_si1 ;
                output      : sm_so ;
                enable_in   : sm_en ;
                enable_out0 : sm_en0 ;
                enable_out1 : sm_en1 ;
            }
        }
    }
}
```

Related Topics

[HostScanInterface/Interface](#)

[Tdr/Interface](#)

[Sib/Interface](#)

Sib

A wrapper that specifies the creation and, optionally, the insertion of a SIB (Segment Insertion Bit) into the IJTAG network.

Usage

```
IjtagNetwork {
    HostScanInterface(id) {
        Sib(id) {
            Interface {
            }
            parent_instance : instance_name ;
            leaf_instance_name : leaf_instance_name ;
            keep_active_during_scan_test : on | off | auto ;
            capture_value : 0 | 1 | self ;
            Attributes {
                attribute_name : attribute_value ;
            }
            Sib(id) {
            }
            Tdr(id) {
            }
            ScanMux(id) {
            }
            DesignInstance(instance_name) {
            }
        }
    }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a SIB (Segment Insertion Bit) into the IJTAG network.

A SIB is a special node in IJTAG that contains a single bit shift register. When the bit is zero, the SIB bypasses the client nodes connected to its host port. When the bit is one, the SIB includes the client nodes connected to its host port as part of the active scan path. [Figure 10-10](#) shows the detailed schematic of its implementation.

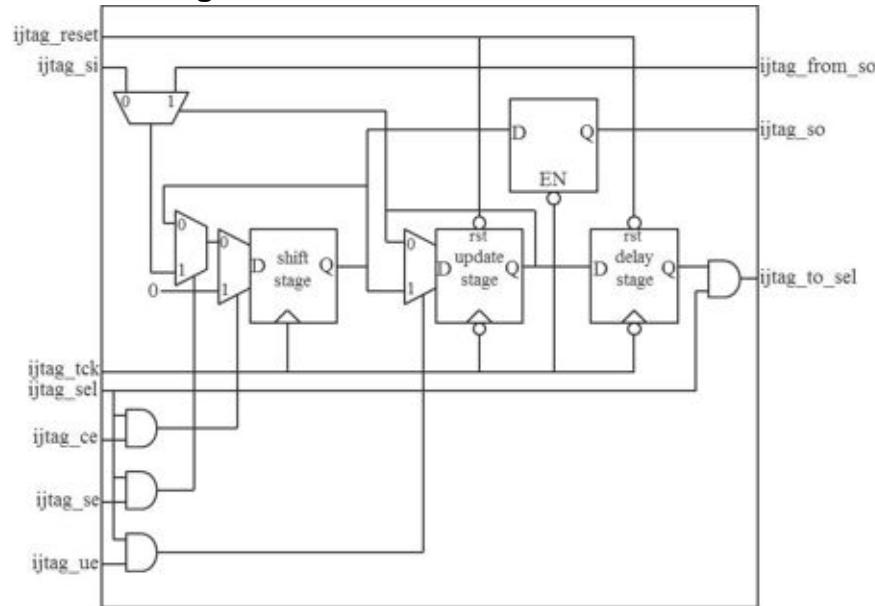
The shift register value is latched into the update stage on the falling edge of TCK when update_en is active. A delay stage is then used to delay the transition of to_select_en by one TCK cycle so as to avoid timing races that will exist if TCK is not perfectly balanced between the SIB and its clients. Refer to “[Update Stage Timing](#)” on page 3277 for more details about this feature.

When the reuse_modules_when_possible property is set or inferred to be on in the **DftSpecification** wrapper, the SIB module names are hardcoded to *design_name_design_id_tessent_sib_#* where # is an integer incrementing from 1. A unique module is created for each combination of port names in the interface wrapper, attributes in the

Attributes wrapper, and the value of the capture_value and keep_active_during_scan_test properties.

When the reuse_modules_when_possible property is set or inferred to be off in the **DftSpecification** wrapper, the SIB module names are hardcoded to *design_name_design_id_tessent_sib_id*.

Figure 10-10. SIB Internal Schematic



As described in the [create_dft_specification](#) command description, the instruments that need to remain active during scan are inserted under one SIB, and the ones which are scan tested, such as a memory BIST controller, are inserted under another SIB. This SIB is identified with a tessent_dft_function attribute set to “scan_tested_instrument_host” and is commonly referred to as the Sib(sti). When this attribute is defined as such, the SIB is equipped with the added logic shown in [Figure 10-11](#) on page 3267 such that the IJTAG network which is scan tested is fully exercised by the scan test. Notice that the Sib(sti) is also equipped with a mini-OCC used to give control to ATPG and LogicBIST as to when the scan tested TCK domain is to receive clock pulses. See [Figure 10-11](#) and [Figure 10-13](#) for a schematic view of the scan isolation logic and the mini-OCC. The ltest_to_en, ltest_to_mem_bypass_en, ltest_to_mcpBounding_en, and ltest_to_se output ports are used during Memory BIST insertion to pre-connect its logic test ports in order to meet logic test design rules (DRCs) and enable memory bypass and RAM sequential testing. The ltest_to_mcpBounding_en port is useful if you want to run logicBist on the memory BIST logic at speed.

The logic test input signals shown on the lower left side of the Sib(sti) in [Figure 10-11](#) are left tied off when the Sib(sti) is inserted. You need to drive those inputs and pick up the sub-chain during scan insertion to achieve high fault coverage of the scan testable portion of the IJTAG network. If you use the [add_dft_signals](#) command to create the scan_en, ltest_en, memory_bypass_en, ltest_to_mcpBounding_en, tck_occ_en and shift_capture_clock DFT signals, the corresponding input ports on the Sib(sti) will automatically be connected to them as

they become available. You will then benefit from all the automation provided by the [set_static_dft_signal_values](#) and [add_core_instances](#) commands to setup the Sti(sti) for logic test and to configure the mini-OCC. The ltest_capture_en port is left tied on, and the ltest_static_control_clock_mode and ltest_clock_sequence ports are left tied off unless the [LogicBist](#) wrapper is used in which case they are driven by the LogicBIST controller.

The mini-OCC enables testing of the scan tested instruments under the STI Sib (for example, MBIST hardware) given that this clock domain is different from the clock domains driving the functional logic and that the test clock driving the STI logic during logic test is usually also the trigger clock for the functional OCCs in fast capture mode (at-speed test). If you forget to add the tck_occ_en DFT signal or if you choose to not activate the TCK OCC during logic test, the TCK OCC will still inject the shift clock into the TCK domain during the load_unload sequence but it will not pulse the TCK domain during the capture sequence. The IJTAG values feeding the functional domains will have known values, but any value observed by the TCK domain will not be captured and will result in “ATPG Untestable” faults.

The [process_dft_specification](#) command creates a TCD for this mini-OCC and writes the TCD into the TSDB as it does with functional clock OCCs created by the DftSpecification/OCC wrappers. And with the functional clock OCCs, you must use the [add_core_instances](#) command in ATPG setup to define this mini-OCC.

Note

 You must add the [tck_occ_en](#) DFT signal otherwise it will not be possible to activate the mini-OCC using the [add_core_instances](#) command. If you do not activate the mini-OCC, then the mini-OCC will still inject a proper shift clock during the shift cycle but will keep the clock off during the capture cycles. The values going from the STI IJTAG network nodes will be known values during logic test, however the STI IJTAG nodes will not be able to capture and observe their inputs, which will result in numerous AU faults in the STI IJTAG network.

Figure 10-11. Sib(sti) With Scan Isolation Chain

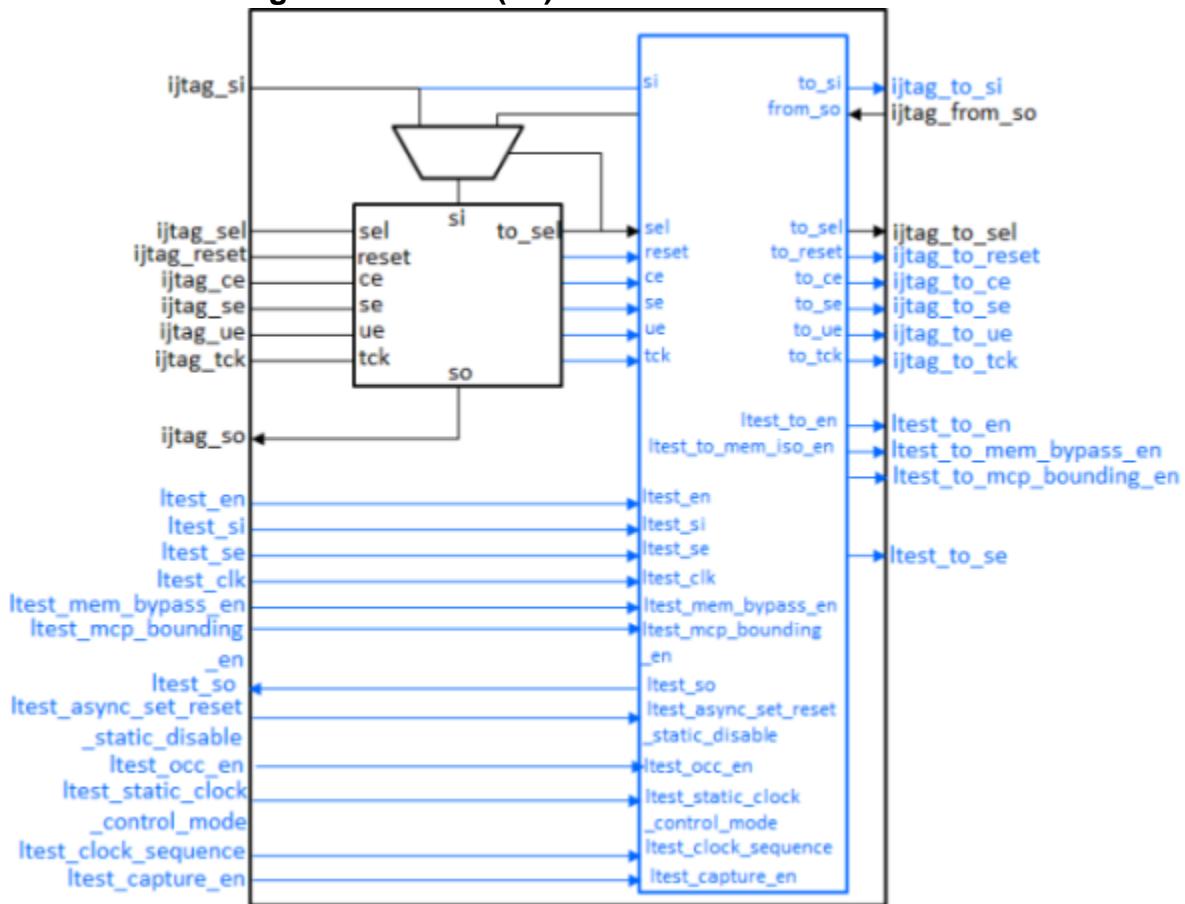


Figure 10-12. Added Scan Isolation Logic for SIB(sti)

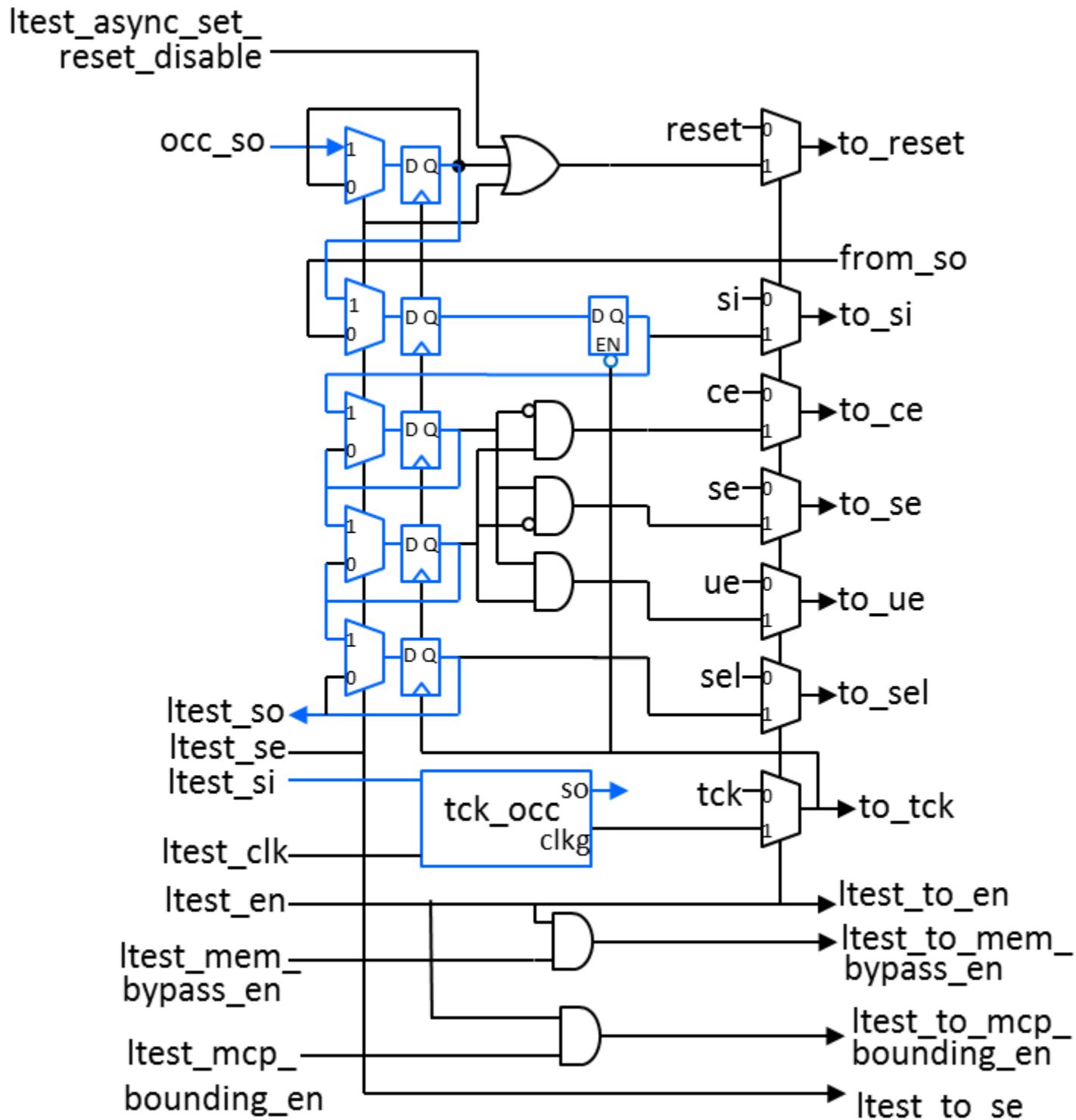
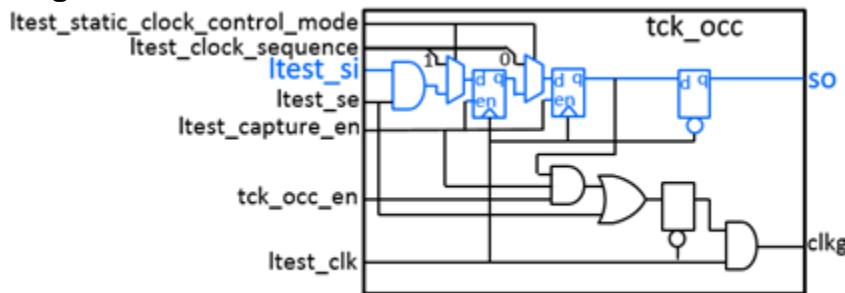


Figure 10-13. Mini-OCC for Scan Tested TCK Domain



Arguments

- *id*
A unique id that contains only letters, numbers, and underscores. Every Sib wrapper must have a different id value inside the [IitagNetwork](#) wrapper.
 - *parent_instance : instance_name ;*
A string property that specifies the parent instance in which to instantiate the SIB module. If unspecified, the SIB is instantiated in the root module. You can use this property to push a SIB down near the client interfaces it interacts with.
If the specified *instance_name* is an instance of a repeated module, a warning is displayed during validation and a *uniquify_instances* command is automatically issued as part of the insertion process to uniquify the *instance_name*.
To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a *DesignInstance* wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the *set_current_design* command in setup mode.
 - *leaf_instance_name : leaf_instance_name ;*
A string property that specifies the leaf instance name of the SIB. It defaults to *design_name_dft_spec_id_tessent_sib_sib_id_inst* when unspecified. The *dft_spec_id* string is the id specified in the [DftSpecification](#) wrapper, and the *sib_id* is the id of the SIB. Tessent Shell checks for conflicts before instantiating the SIB and, if needed, applies the unqualification suffix specified by the [set_insertion_options](#) command.
 - *keep_active_during_scan_test : on | off | auto ;*
A property that specifies that all nodes serially connected to the SIB are to be kept active during scan test modes. For information about when to use this property, see the section “[Scan Testability of the IJTAG Network](#)” on page 3225.
 - *capture_value : 0 | 1 | self ;*
The property controls what the shift register captures during the capture cycle. By default it captures a constant 0 which provides a predictable value that can always be compared when

it comes out on the scan-out pin. Specifying “self” makes the SIB hold during capture. The multiplexer closest to the shift stage in [Figure 10-10](#) is simply removed when using the “self” value. To diagnose a hold time issue in the scan path, it is recommended to keep the default capture value of 0.

- Attributes/*attribute_name* : *attribute_value* ;

A data wrapper that defines arbitrary attributes in the SIB ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The *attribute_name* can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The *attribute_value* can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using [get_attribute_value_list](#), you must have registered the attribute using the “[register_attribute-object_type icl_module](#)” command. For example, assuming you have registered the att1 attribute on the [icl_module](#) object type, and you have added the att1 attribute with a value of “v1” in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the “[get_icl_modules -filter att1==v1](#)” command.

Examples

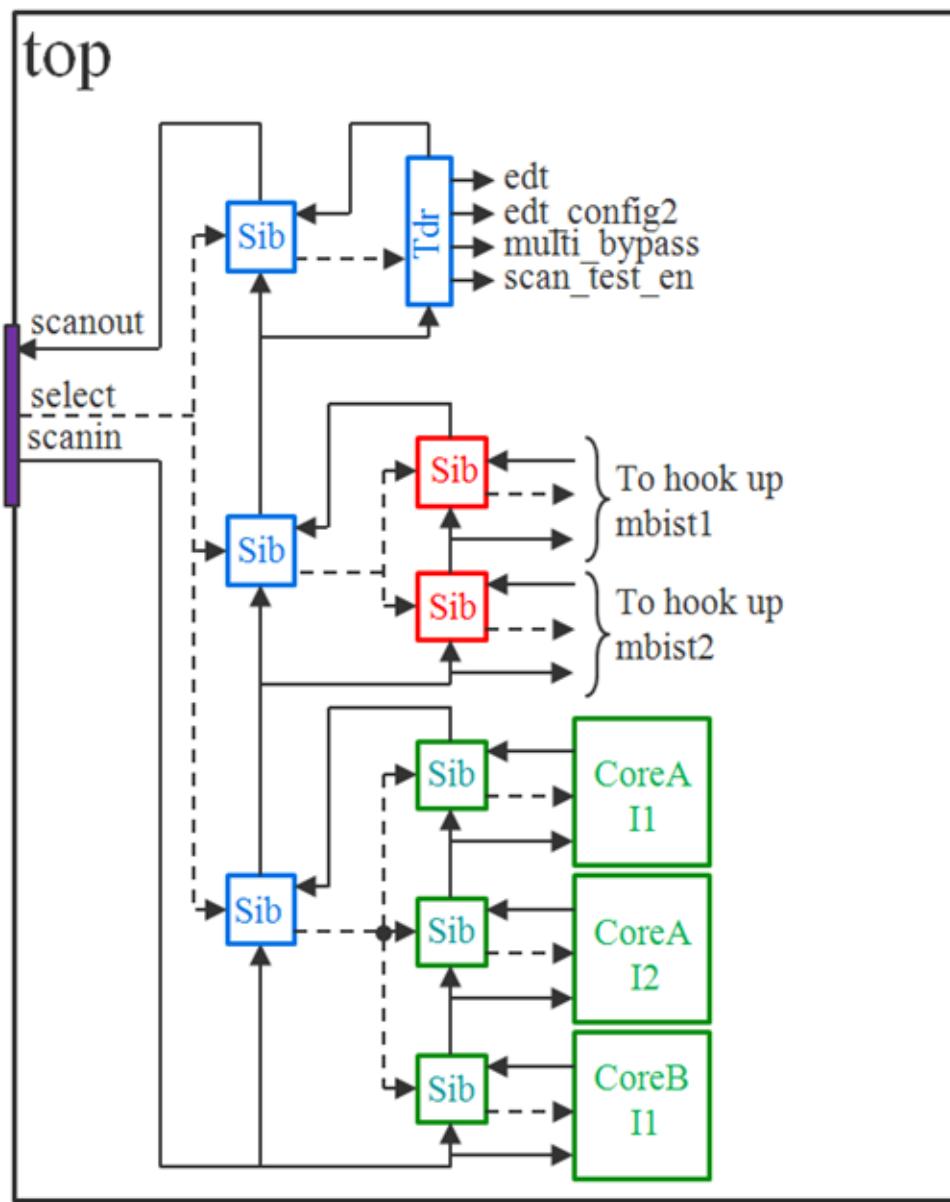
The following example illustrates the use of SIBs to provide discrete access to multiple parts of a complex network. There are three SIBs at the top of the network. The first SIB, which is closest to the scan-out and has an *id* value of “logictest_hw”, is used to provide access to a TDR that is used to create the scan mode enable signals. Because the TDR *id* has a value of “scan_modes”, the property *keep_active_during_scan_test* defaults to on, which in turns propagates to the three SIBs at the top of the network.

The second SIB, with an *id* value of “scannable_instruments”, is used to access all instruments that will not be kept active during scan test but, instead, will be scan tested like the rest of the normal functional logic. This SIB can remain closed during the scan test modes such that the rest of the network remains stable and provides valid and undisturbed *test_setup* values.

The third SIB, with an *id* value of “child_cores”, is used to access a chain of SIBs. The chained SIBs independently open the scan path to each child core. The presence of these SIBs guarantees that a defective core or a powered down core can always be bypassed when accessing other cores.

```
DftSpecification(top,setup_hw) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Sib(logictest_hw) {
                Tdr(scan_modes) {
                    DecodedSignal(edt) {
                        decode_values : 2'b01, 2'b10 ;
                    }
                    DecodedSignal(edt_config2) {
                        decode_values : 2'b10 ;
                    }
                    DecodedSignal(multi_bypass) {
                        decode_values : 2'b11 ;
                    }
                    DecodedSignal(scan_test_en) {
                        decode_values : 2'b01, 2'b10, 2'b11 ;
                    }
                }
            }
        }
        // Everything below this Sib is scan tested.
        // This sibs remains off when setting up logictest_hw
        sib(scannable_instruments) {
            sib(mbist1) {
            }
            sib(mbist2) {
            }
        }
        sib(child_cores) {
            sib(c1) {
                DesignInstance(corea_i1) {
                }
            }
            sib(c2) {
                DesignInstance(corea_i2) {
                }
            }
            sib(c3) {
                DesignInstance(coreb_i1) {
                }
            }
        }
    }
}
```

Figure 10-14. SIB Example Schematic



Related Topics

[DesignInstance](#)

[ScanMux](#)

[IjtagNetwork](#)

[Tdr](#)

Sib/Interface

A wrapper that contains a Client and Host wrapper that together fully specify the client and host ports.

Usage

```
IJtagNetwork {
    HostScanInterface(id) {
        Sib(id) {
            Interface {
                Client {
                    select : port_name ; // default: ijttag_sel
                    tck : port_name ; // default: ijttag_tck
                    scan_in : port_name ; // default: ijttag_si
                    scan_out : port_name ; // default: ijttag_so
                    reset : port_name ; // default: ijttag_reset
                    reset_polarity : active_low | active_high ;
                    shift_en : port_name ; // default: ijttag_se
                    capture_en : port_name ; // default: ijttag_ce
                    update_en : port_name ; // default: ijttag_ue
                }
                Host {
                    select : port_name ; // default: ijttag_to_sel
                    scan_in : port_name ; // default: ijttag_from_so
                }
            }
        }
    }
}
```

Description

A wrapper that contains a Client and Host wrapper that together fully specify the client and host ports.

The Client wrapper provides properties to fully specify the names of the eight client ports, and also contains a property to define the reset polarity of the reset port. The Host wrapper provides properties to fully specify the names of the two host ports.

Arguments

- Client/select : *port_name* ;
This property defines the name of the port labelled “ijtag_sel” in [Figure 10-10](#). The default is “ijtag_sel”.
- Client/tck: *port_name* ;
This property defines the name of the port labelled “ijtag_tck” in [Figure 10-10](#). The default is “ijtag_tck”.
- Client/scan_in : *port_name* ;
This property defines the name of the port labelled “ijtag_si” in [Figure 10-10](#). The default is “ijtag_si”.

- Client/scan_out : *port_name* ;
This property defines the name of the port labelled “ijtag_so” in [Figure 10-10](#). The default is “ijtag_so”.
- Client/reset : *port_name* ;
This property defines the name of the port labelled “ijtag_reset” in [Figure 10-10](#). The default is “ijtag_reset”.
- Client/reset_polarity : active_low | active_high ;
This property defines the active polarity of the port labelled “ijtag_reset” in [Figure 10-10](#). The default is “active_low”.
- Client/shift_en : *port_name* ;
This property defines the name of the port labelled “ijtag_se” in [Figure 10-10](#). The default is “ijtag_se”.
- Client/capture_en : *port_name* ;
This property defines the name of the port labelled “ijtag_ce” in [Figure 10-10](#). The default is “ijtag_ce”.
- Client/update_en : *port_name* ;
This property defines the name of the port labelled “ijtag_ue” in [Figure 10-10](#). The default is “ijtag_ue”.
- Host/select : *port_name* ;
This property defines the name of the port labelled “ijtag_to_sel” in [Figure 10-10](#). The default is “ijtag_to_sel”.
- Host/scan_in : *port_name* ;
This property defines the name of the port labelled “ijtag_from_so” in [Figure 10-10](#). The default is “ijtag_from_so”.

Examples

The following example illustrates fully specifying a unique name for each port.

```
IjtagNetwork {
    HostScanInterface (id) {
        Sib(id) {
            Interface {
                Client {
                    select      : sel ;
                    tck         : tck ;
                    scan_in     : si ;
                    scan_out    : so ;
                    reset       : reset ;
                    shift_en    : se ;
                    capture_en  : ce ;
                    update_en   : ue ;
                }
                Host {
                    select      : to_sel ;
                    scan_in     : from_so ;
                }
            }
        }
    }
}
```

Related Topics

[HostScanInterface/Interface](#)

[Tdr/Interface](#)

[ScanMux/Interface](#)

Tdr

A wrapper that specifies the creation and, optionally, the insertion of a TDR inside the IJTAG network.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tdr(id) {
                DataInPorts {
                }
                DataOutPorts {
                }
                Interface {
                }
                parent_instance           : instance_name ;
                leaf_instance_name       : leaf_instance_name ;
                keep_active_during_scan_test : on | off | auto ;
                length                   : integer ;           // default: auto
                extra_bits_capture_value : 0 | 1 | self ;
                reset_value               : binary ;          // default: auto
                Attributes {
                    attribute_name      : attribute_value ;
                }
                DecodedSignal(signal_name) {
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TDR inside the IJTAG network.

[Figure 10-16](#) on page 3281 shows the detailed schematic of a 3-bit TDR with some data-in and data-out ports as well as some decoded output signals. When requesting data-out ports or decoded signals, the default is to always use an update stage between the shift stage and the output port. In the case of decoded signals, the decoding logic is between the shift stage and the update stage. It is possible to request that the update stage be removed in situations where it does not matter that the data bits toggle during the shift cycles. See the description of the output_timing property found in the [Tdr/DataOutPorts](#) and [Tdr/DecodedSignal](#) sections. On the other hand, if the output data bit is used to enable or disable parts of the IJTAG network, its output timing must be further delayed by an extra cycle to avoid the race condition described in the [“Update Stage Timing”](#) on page 3277. The TCK clock only needs to be balanced within the TDR module but can tolerate as much as 50% of a TCK period between nodes since a negative edge retiming flip-flop is used on the scan-out port of the [Tdr](#) and [Sib](#) nodes.

The TDR module names are hardcoded to *design_name_design_id_tessent_tdr_id*.

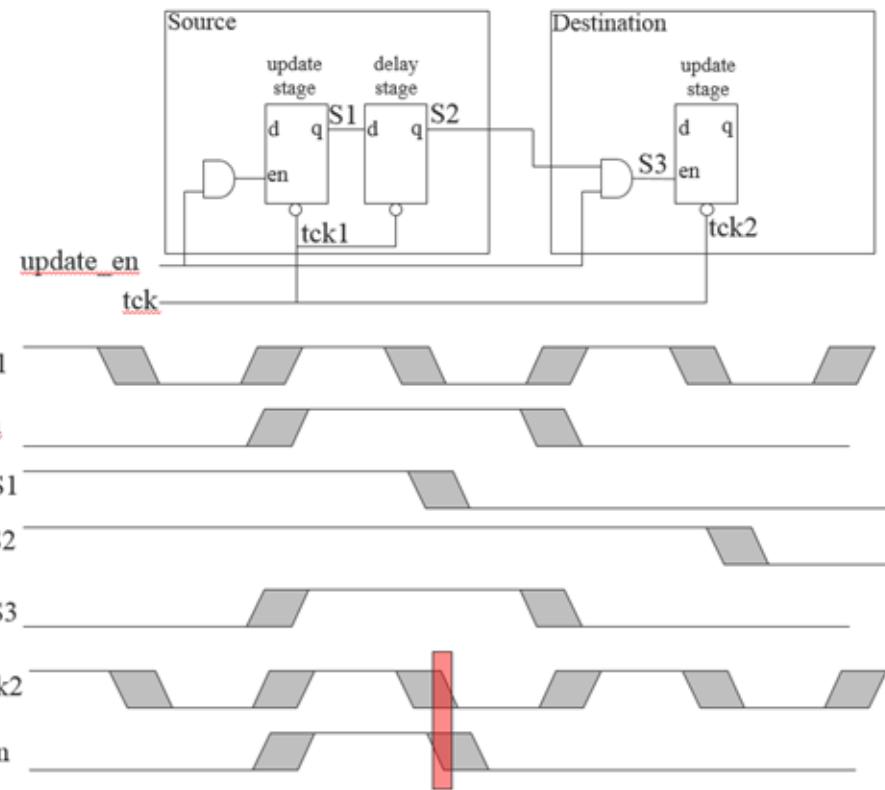
Update Stage Timing

The IJTAG network can be seen as very similar to a 1149.1 or a 1500 network. It has, however, one extra level of freedom where a TDR bit can be used to enable or disable a part of the network while that part of the network is also active. In 1149.1, you never need to worry about races on the select lines because you are in the IR states when changing the instruction and it is only used the next time you go to the DR states. In 1500, the WIR and the selected TDR are accessed in a mutually exclusive manner as controlled by the selectWIR signal.

In IJTAG, you do not have this restriction. If you look at the schematic in [Example 10-9](#) on page 3259, you can see that while you are loading the green TDR, you may also be loading into the black TDR a value that will take out the green TDR for the next scan load and, instead, select the red TDR. As the black, green and red TDRs all receive the same update_en signal, it is important that the select signal reaching the green TDR is still active by the time the falling edge of TCK reaches the green TDR. It is also important that the select signal reaching the red TDR is still inactive by the time the falling edge of TCK reaches the red TDR.

If these conditions are not met, the green TDR may not perform its last update and the red update stage may be prematurely disturbed. This racing condition is illustrated in [Figure 10-15](#). The source box can be seen as the black TDR while the destination box can be seen as the green TDR in [Figure 10-9](#) on page 3259. If you have skew in the TCK clock distribution where TCK2 is delayed with respect to TCK1, the update stage value labeled S1 could change too soon and block the overlap of S1 and update_en at the destination update flip-flop. Because the enabling can be going in all directions, you either have to precisely balance your TCK clock network all over the network or do as is done in Tessent Shell and add an extra delay stage for those select signals to remove the race condition even in the presence of large TCK clock skew.

Figure 10-15. Update Stage Timing Diagram



Arguments

- *id*
A unique id that contains only letters, numbers, and underscores. Every Tdr wrapper must have a different id inside the [IjtagNetwork](#) wrapper.
- *parent_instance* : *instance_name* ;
A string property that specifies the parent instance in which to instantiate the TDR module. When left unspecified, the TDR is instantiated in the common ancestor of the DataInPorts, DataOutPorts, and DecodedSignal connections. You can use this property to push a TDR down into a different location. You can specify “.” as the *parent_instance* to force the TDR to be placed in the root module independent of the DataInPort, DataOutPort, and DecodedSignal connections. When the TDR has no DataInPorts and DataOutPorts connections, the *parent_instance* property defaults to the root module.

If the specified or inferred *instance_name* is an instance of a repeated module, a warning is displayed during validation and a *uniquify_instances* command is automatically issued as part of the insertion process to uniquify the *instance_name*.

To perform IjtagNetwork insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a

single Tessent Shell invocation. You simply change the current design using the `set_current_design` command in setup mode.

- `leaf_instance_name : leaf_instance_name ;`

A string property that specifies the leaf instance name of the TDR. It defaults to `design_name_design_spec_id_tessent_tdr_tdr_id_inst` when unspecified. The `dft_spec_id` string is the id specified in the [DftSpecification](#) wrapper, and the `tdr_id` string is the id of the TDR. Tessent Shell checks for conflicts before instantiating the TDR and, if needed, applies the unification suffix specified by the [set_insertion_options](#) command.

- `keep_active_during_scan_test : on | off | auto ;`

A property that specifies that all nodes serially connected to the TDR are to be kept active during scan test modes. For information about when to use this property, see section “[Scan Testability of the IJTAG Network](#)” on page 3225. This property defaults to on when the id is named “`scan_modes`”.

- `length : integer ;`

A property to request a specific shift register length. By default, the length is determined to be large enough to satisfy the specified DataInPorts/count, DataOutPorts/count or DecodedSignal/decode_values properties.

- `extra_bits_capture_value : 0 | 1 | self ;`

A property that applies to the upper bits of the TDR when the length is larger than the DataInPorts/count value. The specified values control what the extra shift register bits capture during the capture cycle. The value self means that it will hold its value.

- `reset_value : binary ;`

A property that specifies a binary number as wide as the TDR to specify the reset value of the DataOutPorts and the DecodedSignals. The shift register stage is not reset. The ICL description of the TDR automatically sets the DefaultLoadValue of the ScanRegister equal to the specified reset value such that if a TDR bit resets to 1, the DefaultLoadValue for that bit will also be set to 1.

- `Attributes/attribute_name : attribute_value ;`

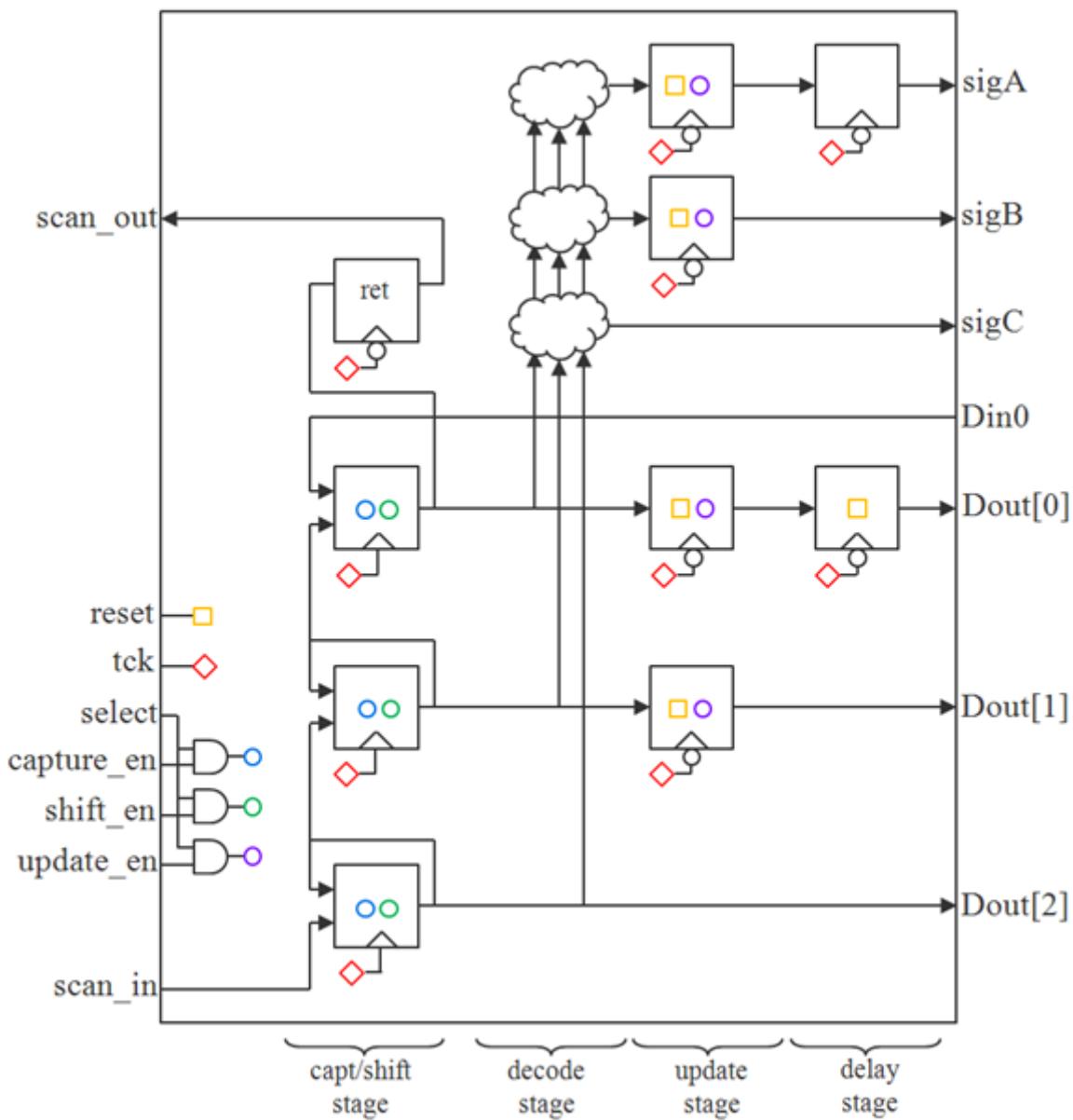
A data wrapper that defines arbitrary attributes in the TDR ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The `attribute_name` can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The `attribute_value` can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using `get_attribute_value_list`, you must have registered the attribute using the “[register_attribute-object_type icl_module](#)” command. For example, assuming you have registered the `att1` attribute on the [icl_module](#) object type, and you have added the `att1` attribute with a value of “`v1`” in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the “[get_icl_modules -filter att1==v1](#)” command.

Examples

The following example illustrates the syntax used to create a TDR exactly like the one shown in [Figure 10-16](#). Notice how the output_timing property is used to make some of the outputs unlatched and some having the extra delay stage needed for ijttag_scan_selection.

```
DftSpecification(MyCore,rtl) {
    IJtagNetwork {
        HostScanInterface(purple) {
            Tdr(blue) {
                length : 3 ;
                extra_bits_capture_value : self ;
                DataOutPorts {
                    count : 3 ;
                    port_naming : Dout[%d] ;
                    output_timing(0) : ijttag_scan_selection ;
                    output_timing(1) : normal ;
                    output_timing(2) : unlatched ;
                }
                DataInPorts {
                    count : 1 ;
                    port_naming : Din%d ;
                }
                DecodedSignal(sigA) {
                    decode_values : 3'bx00 ;
                    output_timing : ijttag_scan_selection ;
                }
                DecodedSignal(sigB) {
                    decode_values : 3'bx01 ;
                    output_timing : normal ;
                }
                DecodedSignal(sigC) {
                    decode_values : 3'bx1x ;
                    output_timing : unlatched ;
                }
            }
        }
    }
}
```

Figure 10-16. Detailed View of TDR Schematic



Related Topics

[IjtagNetwork](#)

[ScanMux](#)

[DesignInstance](#)

[Sib](#)

Tdr/Interface

A wrapper that provides properties to fully specify the names of the eight client ports of the TDR.

Usage

```
IJtagNetwork {
    HostScanInterface(id) {
        Tdr(id) {
            Interface {
                select      : port_name ;      // default: ijttag_sel
                tck         : port_name ;      // default: ijttag_tck
                scan_in     : port_name ;      // default: ijttag_si
                scan_out    : port_name ;      // default: ijttag_so
                reset       : port_name ;      // default: ijttag_reset
                reset_polarity : active_high | active_low;
                shift_en    : port_name ;      // default: ijttag_se
                capture_en  : port_name ;      // default: ijttag_ce
                update_en   : port_name ;      // default: ijttag_ue
            }
        }
    }
}
```

Description

A wrapper that provides properties to fully specify the names of the eight client ports of the TDR.

The wrapper also allows defining the active polarity of the reset port.

Arguments

- **select : *port_name* ;**
This property defines the name of the select port. The default is “ijtag_sel”.
- **tck : *port_name* ;**
This property defines the name of the tck port. The default is “ijtag_tck”.
- **scan_in : *port_name* ;**
This property defines the name of the scan_in port. The default is “ijtag_si”.
- **scan_out : *port_name* ;**
This property defines the name of the scan_out port. The default is “ijtag_so”.
- **reset : *port_name* ;**
This property defines the name of the reset port. The default is “ijtag_reset”.
- **reset_polarity : active_high | active_low ;**
This property defines the active polarity of the reset port. The default is “active_low”.

- `shift_en : port_name ;`
This property defines the name of the shift_en port. The default is “ijtag_se”.
- `capture_en : port_name ;`
This property defines the name of the capture_en port. The default is “ijtag_ce”.
- `update_en : port_name ;`
This property defines the name of the update_en port. The default is “ijtag_ue”.

Examples

The following example illustrates fully specifying a unique name for each port of the TDR.

```
Tdr(id) {
    Interface {
        select      : sel ;
        tck         : tck ;
        scan_in     : si ;
        scan_out    : so ;
        reset       : reset ;
        shift_en    : se ;
        capture_en  : ce ;
        update_en   : ue ;
    }
}
```

Related Topics

[HostScanInterface/Interface](#)

[ScanMux/Interface](#)

[Sib/Interface](#)

Tdr/DataInPorts

A wrapper that specifies the number (count) of data-in ports to create on the TDR, the naming of the ports, and the connections to make to them.

Usage

```
IJtagNetwork {
    HostScanInterface(id) {
        Tdr(id) {
            DataInPorts {
                count           : integer ;          //default: auto
                port_naming    : port_naming,...; //default: ijtag_data_in[%d]
                connection(range) : pin_name;
            }
        }
    }
}
```

Description

A wrapper that specifies the number (count) of data-in ports to create on the TDR, the naming of the ports, and the connections to make to them.

Arguments

- `count : integer ;`

A property that specifies the number of data-in bits needed on the TDR. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the `port_naming` values if they are in a non-indexed format (%d). If the `port_naming` property is set, or defaults to an indexed format, the count is determined to be large enough to satisfy the length of the TDR.

- `port_naming : port_naming, ... ;`

A property that specifies the naming of the data-in ports. The `port_naming` value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

When using the indexed format, you specify `base_name[%d]` when you want to use a bus and you specify `base_name%<integer>d` when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, `Din%2d` will create `Din00`, `Din01`, `Din02`, and so on. `Din%d` and `Din%1d` mean the same thing. The default is “`ijtag_data_in[%d]`”.

When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify “locked”, you will get exactly one data-in port called “locked” as illustrated in [Example 2](#).

- `connection(range) : pin_name ;`

A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sinks, the same bit cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the [process_dft_specification -no_insertion](#) option is not used. If you are using the [process_dft_specification -no_insertion](#) switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of two data-in ports with connections to pins on a design instance.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Sib(orange) {
                Tdr(orange) {
                    DataInPorts {
                        connection(0) : u2/top_green/go ;
                        connection(1) : u2/top_green/done ;
                    }
                }
            }
        }
    }
}
```

Example 2

The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called “locked”. There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

```
DftSpecification(clk_gen,bottom_up) {
    IjtagNetwork {
        HostScanInterface(ijtag) {
            Tdr(1) {
                DataOutPorts {
                    connection(5:3) : pll/mult[2:0] ;
                    connection(2:0) : pll/div[2:0] ;
                    port_naming     : mux_sel, mult[2:0], div[2:0] ;
                }
                DataInPorts {
                    connection(0) : pll/locked ;
                    port_naming   : locked ;
                }
            }
        }
    }
}
```

Related Topics

[Tdr/DataOutPorts](#)

[Tdr/DecodedSignal](#)

Tdr/DataOutPorts

A wrapper that specifies the number (count) of data-out ports to create on the TDR, the naming of the ports, the output_timing of the ports, and the connections to make to them.

Usage

```
IJtagNetwork {
    HostScanInterface(id) {
        Tdr(id) {
            DataOutPorts {
                count           : integer;          // default: auto
                port_naming     : port_naming,...; //default: ijtag_data_out[%d]
                multiplexing    : on | off | auto ;
                output_timing(range) : timing_spec ; // legal : auto unlatched
                                            // normal
                                            // ijtag_scan_selection
                connection(range)   : pin_name ;   // repeatable
            }
        }
    }
}
```

Description

A wrapper that specifies the number (count) of data-out ports to create on the TDR, the naming of the ports, the output_timing of the ports, and the connections to make to them.

Arguments

- count : *integer* ;

A property that specifies the number of data-out bits needed on the TDR. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d). If the port_naming property is set, or defaults to an indexed format, the count is determined to be large enough to satisfy the length of the TDR.

- port_naming : *port_naming*, ... ;

A property that specifies the naming of the data-out ports. The port_naming value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d will create Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly seven

data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in [DataOutPort Connection Multiplexing](#).

- multiplexing : on | off | auto ;

A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See [DataOutPort Connection Multiplexing](#) for illustration.

- output_timing(*range*) : auto | unlatched | normal | ijttag_scan_selection ;

A repeatable property that specifies the output timing of the data-out ports. The default value is auto. The auto will become ijttag_scan_selection if the DataOutPort is used as a ScanMux select source; otherwise, it will become normal.

As you can see in [Figure 10-16](#):

- unlatched — A data-out port can be directly connected to the shift register when output timing is specified as unlatched. This corresponds to Dout[2] in [Figure 10-16](#).
 - normal — A data-out port can be connected to the shift register through an update stage when output timing is specified as normal. This corresponds to Dout[1] in [Figure 10-16](#).
 - ijttag_scan_selection — A data-out port can be connected to the shift register through an update stage and another delay stage when output timing is specified as ijttag_scan_selection. This corresponds to Dout[0] in [Figure 10-16](#). For a description of when the delay stage is needed, refer to “[Update Stage Timing](#)” on page 3277.
- connection(*range*) : pin_name ;
- A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.
- If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the [process_dft_specification -no_insertion](#) option is not used. If you are using the [process_dft_specification -no_insertion](#) switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of four data-out ports with connections to pins on a design instance. The example associated with [Figure 10-16](#) shows the use of the output_timing property.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Sib(orange) {
                Tdr(orange) {
                    DataOutPorts {
                        connection(0) : u2/top_green/en ;
                        connection(3:1) : u2/top_green/mode[2:0] ;
                    }
                }
            }
        }
    }
}
```

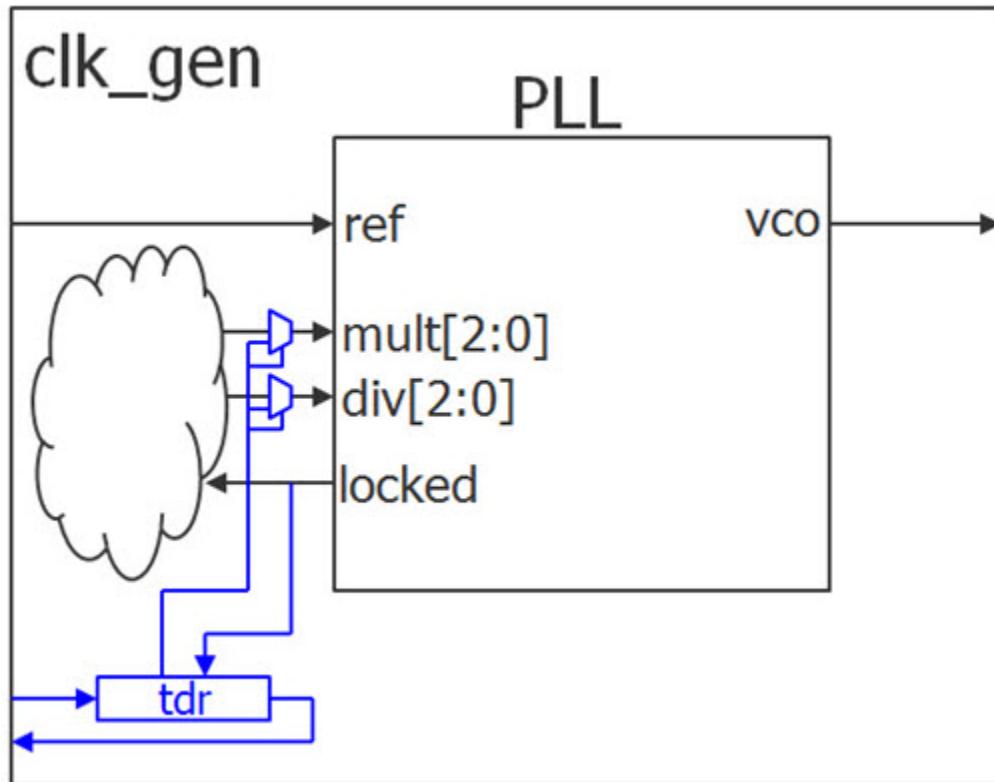
Example 2

The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called “locked”. There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

Multiplexing is left unspecified so it defaults to auto. Because the pll/mult[2:0] and pll/div[2:0] pins have a functional source, a multiplexer is inserted and a 7th data-out bit named mux_sel is used to control the multiplexers.

```
DftSpecification(clk_gen,bottom_up) {
    IjtagNetwork {
        HostScanInterface(ijttag) {
            Tdr(1) {
                DataOutPorts {
                    connection(5:3) : pll/mult[2:0] ;
                    connection(2:0) : pll/div[2:0] ;
                    port_naming : mux_sel, mult[2:0], div[2:0] ;
                }
                DataInPorts {
                    connection(0) : pll/locked ;
                    port_naming : locked ;
                }
            }
        }
    }
}
```

Figure 10-17. DataOutPort Connection Multiplexing



Related Topics

[Tdr/DataInPorts](#)

[Tdr/DecodedSignal](#)

Tdr/DecodedSignal

A wrapper that specifies the creation of a decoded signal.

Usage

```
IjtagNetwork {
    HostScanInterface(id) {
        Tdr(id) {
            DecodedSignal(signal_name) {
                decode_values      : binary_code, ... ;
                multiplexing       : on | off | auto ;
                output_timing     : timing_spec ;
                    // legal : auto unlatched normal
                    //          ijtag_scan_selection
                connection        : port_pin_name ; // repeatable
                Attributes {
                    attribute_name : attribute_value ;
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the creation of a decoded signal.

The decode_values property is used to specify which values of the TDR make the decoded signal active. The connection property specifies connections from the decoded signals.

Arguments

- decode_values : *binary_code*, ... ;

A property that specifies a list of binary codes that define when the decoded signal is active. The binary codes must be as wide as the TDR. The binary numbers may contain Xs as long as there is at least one 0 or one 1 in each value.

- multiplexing : on | off | auto ;

A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added on the TDR to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See [DataOutPort Connection Multiplexing](#) of the [Tdr/DataOutPorts](#) section for an illustration of this.

- output_timing : auto | unlatched | normal | *ijtag_scan_selection* ;

A property that specifies the output timing of the decoded signal port. The default value is auto. The auto will become *ijtag_scan_selection* if the DecodedSignal is used as a ScanMux select source; otherwise, it will become normal.

As you can see in [Figure 10-16](#):

- unlatched — A decoded signal is directly connected to the decoding logic when output timing is specified as unlatched; this corresponds to SigC in [Figure 10-16](#).
- normal — A data-out port is connected to the decoding logic through an update stage when output timing is specified as normal; this corresponds to SigB in [Figure 10-16](#).
- *ijtag_scan_selection* — A data-out port is connected to the decoding logic through an update stage and another delay stage when output timing is specified as *ijtag_scan_selection*; this corresponds to SigA in [Figure 10-16](#).

Refer to the “[Update Stage Timing](#)” on page 3277 for a description of when the delay stage is needed.

- **connection : *port_pin_name* ;**

A repeatable property that specifies connections to make between the decoded signal and existing pins inside the design. Because decoded signals are sources, they can be connected to multiple destinations by simply repeating the connection property.

- **Attributes/*attribute_name* : *attribute_value* ;**

A data wrapper that defines arbitrary attributes on the ICL port associated with the decoded signal. This mechanism is useful for finding special attributed ports in the concatenated ICL. The *attribute_name* can be any identifier starting with a letter and followed by any number of letters, numbers and underscores. The *attribute_value* can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using [get_attribute_value_list](#), you must have registered the attribute using the “[register_attribute-object_type icl_module](#)” command. For example, assuming you have registered the att1 attribute on the [icl_port](#) object type, and you have added the att1 attribute with a value of “v1” to some DecodedSignal ports of your TDR modules using the above Attributes wrapper, you can later quickly find those ports using the “[get_icl_ports -of module_name -filter att1==v1](#)” command. The [icl_port](#) attributes are also inherited by the [icl_pin](#) objects so you can also find the pins using “[get_icl_pins -filter att1==v1](#)”.

Examples

The following example illustrates the creation of three decoded signals with each having a different output timing. The example corresponds to the schematic shown in [Figure 10-16](#).

```
DftSpecification(MyCore,rtl) {
    IJtagNetwork {
        HostScanInterface(purple) {
            Tdr(blue) {
                DecodedSignal(sigA) {
                    decode_values      : 3'bx00 ;
                    output_timing     : ijtag_scan_selection ;
                }
                DecodedSignal(sigB) {
                    decode_values      : 3'bx01 ;
                    output_timing     : normal ;
                }
                DecodedSignal(sigC) {
                    decode_values      : 3'bx1x ;
                    output_timing     : unlatched ;
                }
            }
        }
    }
}
```

Related Topics

[Tdr/DataInPorts](#)

[Tdr/DataOutPorts](#)

Tap

A wrapper that specifies the creation and, optionally, the insertion of a TAP controller into the IJtagNetwork.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                Interface {
                }
                parent_instance           : instance_name ;
                leaf_instance_name       : leaf_instance_name ;
                keep_active_during_scan_test : on | off | auto ;
                bypass_instruction_codes   : auto | binary , ... ;
                add_buffers_on_jtag_signal_sources : on | off | auto ;
                Attributes {
                    attribute_name : attribute_value;
                }
                DataOutPorts {
                }
                HostBscan {
                }
                DeviceIDRegister {
                }
                HostIjtag(id) {
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TAP controller into the IJtagNetwork.

A TAP is a special node in IJTAG. It contains an IEEE1149.1 state machine controlled by a TMS port. The TMS port is used to steer the state machine into various states and control when the capture, shift, and update cycles happen. Unlike a SIB and a TDR, which are only active in the DR States, the TAP has an instruction register that is activated in the IR States. The value in the instruction register determines which data register is selected when in the DR States. The TAP state machine is illustrated in [Figure 10-18](#). A block diagram of a TAP controller is shown in [Figure 10-19](#).

A TAP wrapper is only allowed to exist in a HostScanInterface or inside the Input wrapper of a ScanMux. A TAP is only allowed to be in series with other TAP controllers and can never be in series with [Tdr](#) and [Sib](#) nodes even through [ScanMux](#) nodes. The name of all ports on the TAP controller are user-specifiable using the Interface wrappers. BoundaryScan host ports are present when the [Tap/HostBScan](#) wrapper is specified. The device id register is present when the [Tap/DeviceIDRegister](#) wrapper is specified. Finally, the IJTAG host ports are present when

one or more HostIjtag wrappers are specified. The bypass register is required and always present.

If you are familiar with the LogicVision TAP, you will remember that it had an internal DR register used to observe status bits and control user bits and decoded signals. If you need such a register, you can and should use a [Tdr](#) wrapper inside a HostIjtag wrapper of the Tap; this will give you access to the [Tdr/DataInPorts](#), [Tdr/DataOutPorts](#), and [Tdr/DecodedSignal](#) wrapper syntax.

All registers operate on the rising edge of TCK except for the update stage, labelled “inst_latch” in [Figure 10-19](#), which operates on the falling edge of TCK.

The retiming latch shown in [Figure 10-19](#) is replaced by a retiming flip-flop when the scan_path_retimining property within the IjtagNetwork/ImplementationOptions section is set to “flop”. For the reason explained in the description of that property, the retiming flop is moved as compared to where the latch is shown so as not to be in the path between the from_hostX_so ports and the tdo port in order to avoid the cascaded retiming flip-flops situation described there. As stated in that section, make sure third party nodes include a retiming element on their scan_out when you connect them directly to the scan_in port of the HostIjtag scan interfaces Tesson Tap and you specify the scan_path_retimining property to “flop”, otherwise your elements will not be 1149.1 compliant and will likely not be usable within a board environment.

The tap module names are hardcoded to *design_name_design_id_tesson_tap_id*.

Figure 10-18. TAP FMS Diagram

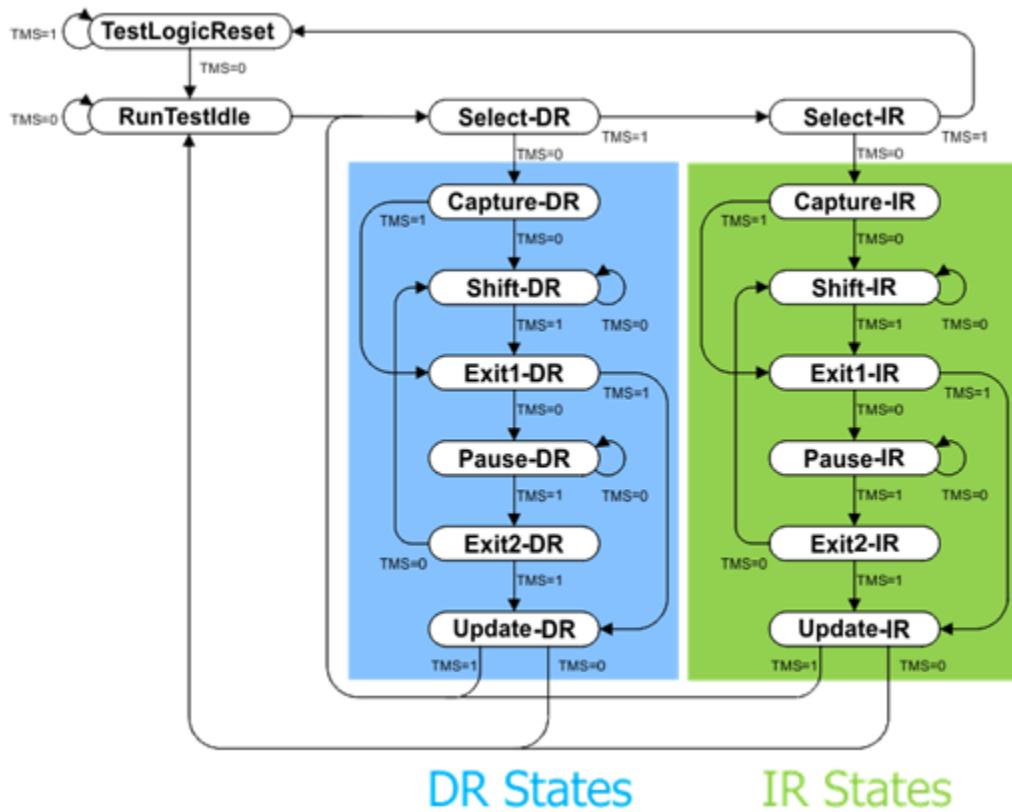
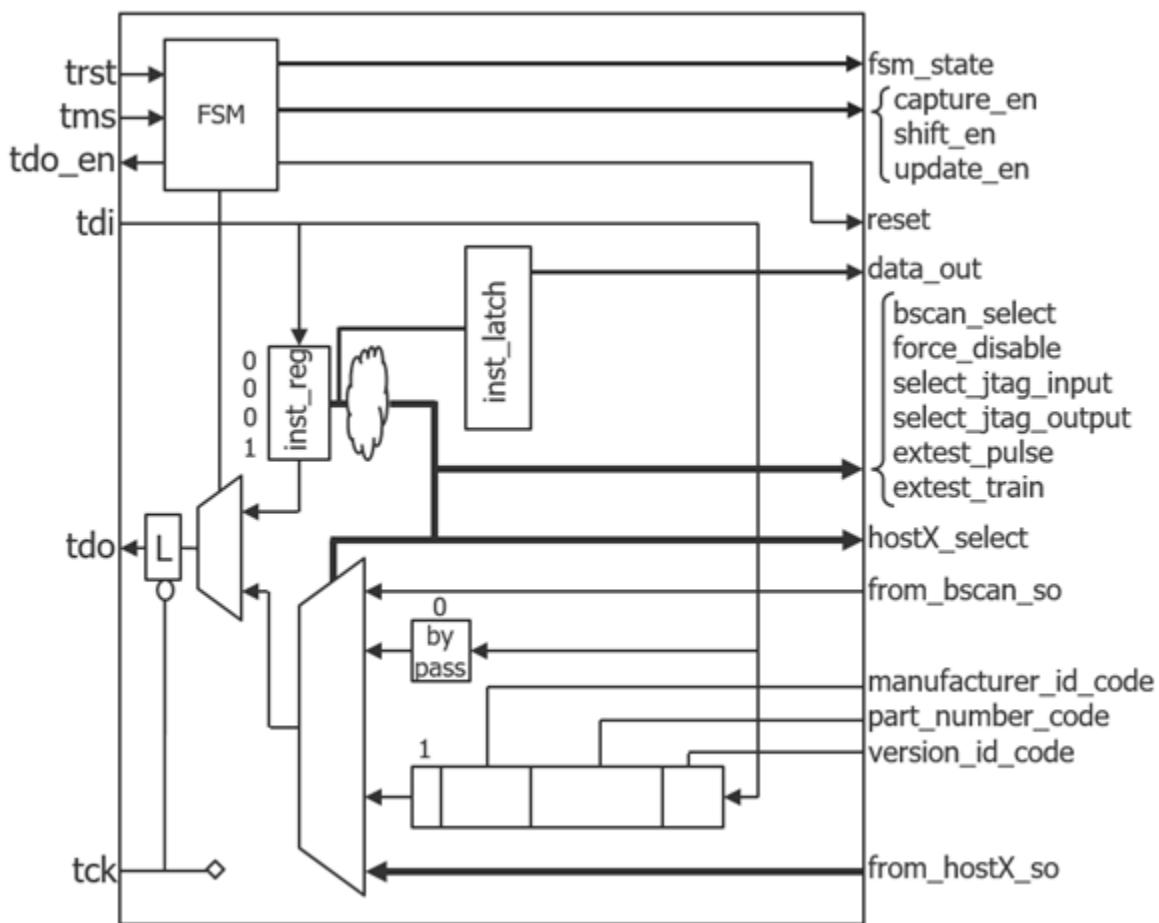


Figure 10-19. TAP Block Diagram

Arguments

- *id*
A unique id that contains only letters, numbers, and underscores. Every Tap wrapper must have a different id value inside the [IJtagNetwork](#) wrapper.
- *parent_instance* : *instance_name* ;
A string property that specifies the parent instance in which to instantiate the TAP module. If unspecified, the TAP is instantiated in the root module. You can use this property to push a TAP down near the client interfaces it interacts with.
If the specified *instance_name* is an instance of a repeated module, the tool issues a warning during validation and automatically issues a [uniquify_instances](#) command as part of the insertion process to uniquify the *instance_name*.
To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a

single Tesson Shell invocation. You simply change the current design using the `set_current_design` command in setup mode.

- `leaf_instance_name : leaf_instance_name ;`

A string property that specifies the leaf instance name of the TAP. It defaults to `design_name_dft_spec_id_tesson_tap_tap_id_inst` when unspecified. The `dft_spec_id` string is the id specified in the [DftSpecification](#) wrapper, and the `tap_id` is the `id` of the Tap wrapper. Tesson Shell checks for conflicts before instantiating the TAP controller and, if needed, applies the unification suffix specified by the [set_insertion_options](#) command.

- `keep_active_during_scan_test : on | off | auto ;`

A property that specifies that all nodes serially connected to the TAP are to be kept active during scan test modes. For information about when to use this property, see the section “[Scan Testability of the IJTAG Network](#)” on page 3225.

- `bypass_instruction_codes : auto | binary, ... ;`

A property used to define one or many opcodes for the BYPASS instruction. The all 0 code is automatically assigned to the BYPASS instruction whether specified or not. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- `add_buffers_on_jtag_signal_sources : on | off | auto ;`

A property that adds persistent buffers to the TAP controller's input control signals: TDI, TMS, TCK and TRST. The buffers are added to create an interception point for the TAP and the nodes in the networks it hosts. When set to “auto”, the buffers will be added if there is at least one InSystemTest controller wrapper in the current DftSpecification. Use this property if you intend to insert InSystemTest controllers to intercept a specific TAP controller.

- `Attributes/attribute_name : attribute_value ;`

A data wrapper that defines arbitrary attributes in the TAP ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL at a later time. The `attribute_name` can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The `attribute_value` can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using `get_attribute_value_list`, you must have registered the attribute using the “[register_attribute-object_type icl_module](#)” command. For example, assuming you have registered the `att1` attribute on the [icl_module](#) object type, and you have added the `att1` attribute with a value of “`v1`” in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the “[get_icl_modules -filter att1==v1](#)” command.

Examples

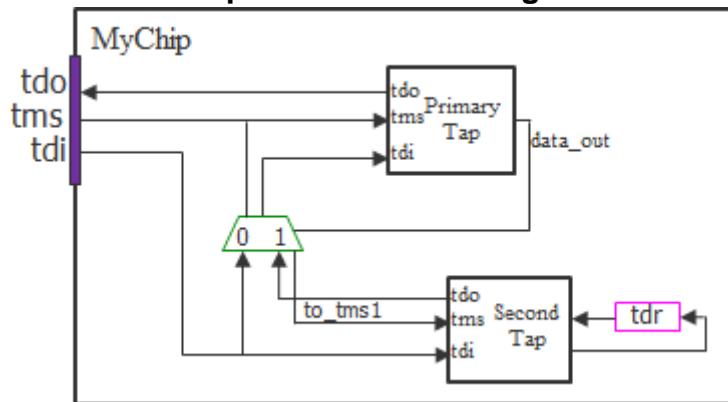
Example 1

This example defines two TAP controllers where a DataOutPort on the primary TAP is used to control if the secondary TAP is active or not. When the DataOutPort on the primary TAP is high, the secondary TAP is in series with it. When the DataOutPort on the primary TAP is low, the secondary TAP is bypassed and it is held in run_test_idle by gating its tms port low.

```

DftSpecification(module_name,id) {
    IJtagNetwork {
        HostScanInterface(purple) {
            Tap(main) {
                DataOutPorts {
                    count : 1 ;
                }
            }
            ScanMux(green) {
                select : Tap(main)/DataOut(0) ;
                Input(1) {
                    Tap(secondary) {
                        HostIJtag(0) {
                            Tdr(pink) {
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Figure 10-20. Example Multi-TAP Configuration Schematic**Example 2**

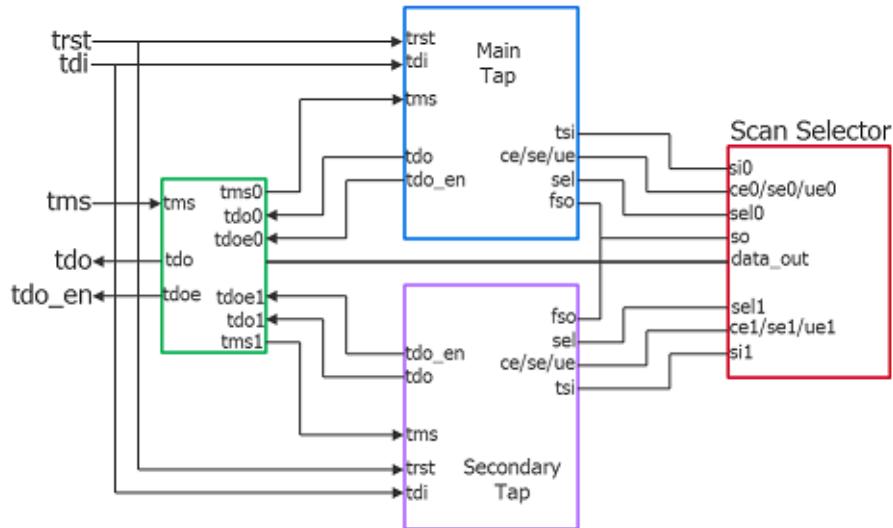
The following example uses a special one bit TDR to select between two TAPs. The special aspect of this circuit is that the one bit TDR (called Scan Selector in Figure 10-21) used to select which TAP is active remains active below the selected TAP. Both TAPs have a HostIJtag connected to one ScanInterface of the Scan Selector and the state of the Scan Selector itself decides which ScanInterface controls it. The advantage of this circuit is that the currently selected TAP remains alone between TDI and TDO. Using the instruction on the Main TAP that accesses the TDR enables changing the value in the TDR to give control to the Secondary TAP. The same method in the newly selected Secondary TAP can be used to return control to the Main TAP. The IJTAG solver can automatically switch between TAPs based on the elements it is being asked to access.

Figure 10-21 shows the block diagram of the circuit. Below it is the DftSpecification used to insert two TAP controllers and a ScanMux and connect the HostIJtag of each TAP to the two

ScanInterfaces of the Scan Selector. The Verilog description and ICL of the Scan Selector are shown next.

The TDR in the Scan Selector module resets to 0 when the trst port is pulled low or the 5 tms sequence is applied for the active TAP. The process_dft_specification.post_insertion callback shown below is used to insert the AND gate during the [process_dft_specification](#) execution. This AND gate is not described in the ICL but it does not matter as it is the implied behavior.

Figure 10-21. Example of TDR Scan Selector



DftSpecification:

```
DftSpecification(chip,rtl) {
    IjtagNetwork {
        HostScanInterface(tap) {
            Interface {
                tck : tck ;
                tdi : tdi ;
                tdo : tdo ;
                trst : trst ;
                tms : tms ;
            }
            ScanMux(tap_mux) {
                select : DesignInstance(scan_selector)/data_out ;
                Input(0) {
                    Tap(main) {
                        HostIjtag(0) {
                            DesignInstance(scan_selector) {
                                scan_interface : Int0 ;
                            }
                        }
                        HostIjtag(1) {
                            tdr(1) {
                                DataOutPorts {
                                    count : 8 ;
                                }
                            }
                        }
                    }
                }
                Input(1) {
                    Tap(secondary) {
                        HostIjtag(0) {
                            DesignInstance(scan_selector) {
                                scan_interface : Int1 ;
                            }
                        }
                        HostIjtag(1) {
                            tdr(2) {
                                DataOutPorts {
                                    count : 8 ;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

The process `_dft_specification.post_insertion` callback to AND the two rstn pins of the TAP:

```
proc process_dft_specification.post_insertion {root_wrapper args} { \
    create_connections chip_rtl_tessent_tap_main_inst/test_logic_reset/ \
    scan_selector/rstn \
    intercept_connection scan_selector/rstn -cell_function_name and \
    -input2 chip_rtl_tessent_tap_secondary_inst/test_logic_reset
}
```

Verilog view of the special TDR called `scan_selector`:

```
module scan_selector (tck, rstn1, sel1, sil, sel1, ue1, so1,
                      rstn0, sel0, si0, se0, ue0, so0, data_out);
  input tck, rstn1, sel1, sil, sel1, ue1, rstn0, sel0, si0, se0, ue0;
  output so1, so0, data_out;
  reg shift_stage, update_stage;
  assign se = (data_out) ? sel1 : se0;
  assign ue = (data_out) ? ue1 : ue0;
  assign sel = (data_out) ? sel1 : sel0;
  assign si = (data_out) ? sil : si0;
  assign rstn = (data_out) ? rstn1 : rstn0;
  assign data_out = update_stage;
  assign so1 = shift_stage;
  assign so0 = shift_stage;
  assign se_gated = sel & se;
  assign ue_gated = sel & ue;
  always @ (posedge tck or negedge rstn) begin
    if (~rstn) begin
      update_stage <= 1'b0;
    end else begin
      if (ue_gated) begin
        update_stage <= shift_stage;
      end
    end
  end
  endalways @ (posedge tck) begin
    if (se_gated) begin
      shift_stage <= si;
    end
  end
end
endmodule
```

ICL view of the special TDR called scan_selector:

```
Module scan_selector {
    TCKPort      tck;
    SelectPort   sel1;
    ScanInPort   si1;
    ShiftEnPort  sel1;
    UpdateEnPort ue1;
    SelectPort   sel0;
    ScanInPort   si0;
    ShiftEnPort  se0;
    UpdateEnPort ue0;
    ScanOutPort  so1 { Source tdr; }
    ScanOutPort  so0 { Source tdr; }
    DataOutPort  data_out { Source tdr; }
    ScanInterface Int1 { Port tck; Port sel1; Port si1;
                         Port sel1; Port ue1; Port so1; }
    ScanInterface Int0 { Port tck;  Port sel0; Port si0;
                         Port se0; Port ue0; Port so0; }
    ScanMux M1 SelectedBy tdr {
        1'b0 : si0;
        1'b1 : si1;
    }
    ScanRegister tdr {
        ScanInSource M1;
        ResetValue 1'b0;
    }
}
```

Tap/Interface

A wrapper that contains a Client and Host wrapper that together fully specify the names and polarity of the client and host ports.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                Interface {
                    fsm_state : port_name ; // def: fsm_state[%d]
                    Client {
                        tdi : port_name ; // default: tdi
                        tdo : port_name ; // default: tdo
                        tck : port_name ; // default: tck
                        tms : port_name ; // default: tms
                        trst : port_name ; // default: trst
                        tdo_en : port_name ; // default: tdo_en
                        tdo_en_polarity : active_high | active_low | auto ;
                    }
                    Host {
                        reset : port_name ; // default: test_logic_reset
                        capture_en : port_name ; // default: capture_dr_en
                        shift_en : port_name ; // default: shift_dr_en
                        update_en : port_name ; // default: update_dr_en
                        reset_polarity : active_high | active low ;
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that contains a Client and Host wrapper that together fully specify the names and polarity of the client and host ports.

The Client wrapper provides properties to fully specify the names of the six client ports, and also contains a property to define the polarity of the tdo_en port. The Host wrapper provides properties to fully specify the names of the four global host ports as well as a property to define the polarity of the reset port. The rest of the host ports related to the IJTAG and boundary scan ports are specified in the [Tap/HostIjtag](#) and [Tap/HostBScan](#) wrappers respectively.

Arguments

- fsm_state : *port_name* ;

This property defines the name of the port labeled “fsm_state” in [Figure 10-19](#). The specified value must end with “[%d]”. The default is “fsm_state[%d]”.

- Client/td : *port_name*;
This property defines the name of the port labeled “td” in [Figure 10-19](#). The default name when unspecified is “td”.
- Client/tdo : *port_name*;
This property defines the name of the port labeled “tdo” in [Figure 10-19](#). The default name when unspecified is “tdo”.
- Client/tck : *port_name*;
This property defines the name of the port labeled “tck” in [Figure 10-19](#). The default name when unspecified is “tck”.
- Client/tms : *port_name*;
This property defines the name of the port labeled “tms” in [Figure 10-19](#). The default name when unspecified is “tms”.
- Client/trst : *port_name*;
This property defines the name of the port labeled “trst” in [Figure 10-19](#). The default name when unspecified is “trst”. This port is always active low.
- Client/tdo_en : *port_name*;
This property defines the name of the port labeled “tdo_en” in [Figure 10-19](#). The default name when unspecified is “tdo_en”. Its active polarity is controlled by the tdo_en_polarity property. The port is active in the Shift-DR and Shift-IR states. See [Figure 10-18](#) for a diagram of the state machines.
- Client/tdo_en_polarity : active_high | active_low | auto ;
This property controls the polarity of the tdo_en port. The default is active_low. When the **DftSpecification** is created by the [create_dft_specification](#) command, the property is automatically specified to match the polarity of the enable pin for the output pad buffer associated with the tdo port.
- Host/reset : *port_name* ;
This property defines the name of the port labelled “reset” in [Figure 10-19](#). The default name when unspecified is “test_logic_reset”. Its active polarity is defined using the reset_polarity property. This port is active when the TAP fsm is in the TestLogicReset state which can be synchronously entered by keeping the tms port high for a minimum of 5 TCK cycles or asynchronously entered by setting the trst port to zero.
- Host/capture_en : *port_name* ;
This property defines the name of the port labeled “capture_en” in [Figure 10-19](#). The default name when unspecified is “capture_dr_en”.
- Host/shift_en : *port_name* ;
This property defines the name of the port labelled “shift_en” in [Figure 10-19](#). The default name when unspecified is “shift_dr_en”.

- Host/update_en : *port_name* ;

This property defines the name of the port labelled “update_en” in Figure 10-19. The default name when unspecified is “update_dr_en”.

- Host/reset_polarity : active_high | active_low ;

This property defines the active polarity of the reset port.

Examples

This example defines all client ports with a “client_” prefix and all host ports with a “host_” prefix. The polarity of tdo_en and reset are both specified as active high.

```
DftSpecification(module_name,id) {  IjtagNetwork {
    HostScanInterface(id) {
        Tap(id) {
            Interface {
                Client {
                    tdi           : client_si ;
                    tdo           : client_so ;
                    tck           : client_tck ;
                    tms           : client_tms ;
                    trst          : client_reset ;
                    tdo_en        : client_so_en ;
                    tdo_en_polarity : active_high ;
                }
                Host {
                    reset          : host_reset ;
                    capture_en     : host_capture_en ;
                    shift_en       : host_shift_en ;
                    update_en      : host_update_en ;
                    reset_polarity : active_high ;
                }
            }
        }
    }
}
```

Tap/HostBScan

The presence of this wrapper specifies that the TAP is to be equipped with a port capable of hosting a boundary scan chain.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                HostBscan {
                    Interface {
                        select : port_name ; // def: host_bscan_to_sel
                        force_disable : port_name ; // def: force_disable
                        select_jtag_input : port_name ; // def: select_jtag_input
                        select_jtag_output : port_name ; // def: select_jtag_output
                        extest_pulse : port_name ; // def: extest_pulse
                        extest_train : port_name ; // def: extest_train
                        scan_in : port_name ; // def: host_bscan_from_so
                    }
                    InstructionCodes {
                        CLAMP : auto | unused | binary, ... ;
                        EXTEST : auto | binary, ... ;
                        EXTEST_PULSE : auto | unused | binary, ... ;
                        EXTEST_TRAIN : auto | unused | binary, ... ;
                        INTEST : auto | unused | binary, ... ;
                        SAMPLE_PRELOAD : auto | binary, ... ;
                        HIGHZ : auto | unused | binary, ... ;
                    }
                }
            }
        }
    }
}
```

Description

The presence of this wrapper specifies that the TAP is to be equipped with a port capable of hosting a boundary scan chain.

The interface wrapper is used to change the default port naming of the various functions. When the HostBscan wrapper is present, the ICL description of the Tap includes all the elements described in the section “[Requirements on a TAP to be usable for BoundaryScan](#)” on page 3385 on page 92.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Arguments

- Interface/select : *port_name* ;

This property defines the name of the port labelled “bscan_select” in [Figure 10-19](#). The default is “host_bscan_to_sel”.

- Interface/force_disable : *port_name* ;

This property defines the name of the port labelled “force_disable” in [Figure 10-19](#). The default is “force_disable”.

- Interface/select_jtag_input : *port_name* ;

This property defines the name of the port labelled “select_jtag_input” in [Figure 10-19](#). The default is “select_jtag_input”.

- Interface/select_jtag_output : *port_name* ;

This property defines the name of the port labelled “select_jtag_output” in [Figure 10-19](#). The default is “select_jtag_output”.

- Interface/extest_pulse : *port_name* ;

This property defines the name of the port labelled “extest_pulse” in [Figure 10-19](#). The default is “extest_pulse”. This port is not created when the EXTEST_PULSE property inside the [Tap/HostIjtag](#) wrapper is specified as “unused”.

- Interface/extest_train : *port_name* ;

This property defines the name of the port labelled “extest_train” in [Figure 10-19](#). The default is “extest_train”. This port is not created when the EXTEST_TRAIN property inside the [Tap/HostIjtag](#) wrapper is specified as “unused”.

- Interface/scan_in : *port_name* ;

This property defines the name of the port labelled “from_bscan_so” in [Figure 10-19](#). The default is “host_bscan_from_so”.

- InstructionCodes/CLAMP : auto | unused | binary, ... ;

A property used to define one or many opcodes for the CLAMP instruction. All codes must differ from the other codes by at least one non-X bit. When left as auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST : auto | binary, ... ;

A property used to define one or many opcodes for the EXTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified.

- InstructionCodes/EXTEST_PULSE : auto | unused | binary, ... ;

A property used to define one or many opcodes for the EXTEST_PULSE instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST_TRAIN : auto | unused | binary, ... ;

A property used to define one or many opcodes for the EXTEST_TRAIN instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/INTEST : auto | unused | binary, ... ;

A property used to define one or many opcodes for the INTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/SAMPLE_PRELOAD : auto | binary, ... ;

A property used to define one or many opcodes for the SAMPLE_PRELOAD instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified.

- InstructionCodes/HIGHZ : auto | unused | binary, ... ;

A property used to define one or many opcodes for the HIGHZ instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the [Tap/HostBScan](#) wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

Examples

This example specifies that the TAP is to be equipped with a boundary scan host port. The port naming is left unspecified so the default naming is used. It specifies explicit opcodes for the required instructions related to boundary scan and makes a few of the optional ones unused. Notice that the all 0 opcode is not used as it is reserved for the bypass register. This example assumes no Tap/HostIjtag and no [Tap/DataOutPorts](#) wrappers are specified as there would be no room for those opcodes.

```
DftSpecification(mychip,rtl) {
    IjtagNetwork {
        HostScanInterface(tap) {
            Tap(main) {
                bypass_instruction_codes : 2'b00 ;
            HostBscan {
                InstructionCodes {
                    CLAMP          : unused ;
                    EXTEST         : 2'b01 ;
                    EXTEST_PULSE   : unused ;
                    EXTEST_TRAIN   : unused ;
                    INTEST         : unused ;
                    SAMPLE_PRELOAD : 2'b11 ;
                    HIGHZ          : unused ;
                }
            }
        }
    }
}
```

Tap/DataOutPorts

A wrapper that specifies the number (count) of data-out ports to create on the TAP module, the naming of the ports, the connections to make from them, and whether multiplexing is to be inserted when making the connections.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                DataOutPorts {
                    count           : integer ;          //default: auto
                    port_naming     : port_naming, ;      //default: user_ir_bits[%d]
                    multiplexing    : on | off | auto ;
                    connection(range) : pin_name ;       // repeatable
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the number (count) of data-out ports to create on the TAP module, the naming of the ports, the connections to make from them, and whether multiplexing is to be inserted when making the connections.

Mentor Graphics does not recommend using the TAP as a source of DataOut ports. Instead, you should use the [Tdr/DataOutPorts](#) wrapper as a source. It is good practice to include a data-out port on a TAP to control the select of a [ScanMux](#) that inserts one or many secondary TAP controllers in series with the main one as illustrated in [Figure 10-20](#). Notice that the property `output_timing` which exists in the [Tdr/DataOutPorts](#) wrapper is not present. There is no risk of races here because the TAP data-out ports change in the Update-IR state while all client registers are only reacting to the Update-DR state. When using the data-out port to control the insertion or exclusion of a TAP controller, the TMS signal is clocked on the rising edge of TCK, and the data-out signal changes on the falling edge of TCK thereby removing any risk of races.

Important Note: The bits in the instruction register and, consequently, the value on the ports is overwritten by the following:

- A TAP reset tested in the `test_logic_reset` test in the [PatternsSpecification/Patterns/TestStep/BoundaryScan/RunTest](#) wrapper. The value of the IR register is the determined by the implemented reset value in the hardware.

You can use a separate [TestStep](#) wrapper for the `test_logic_reset` test in which the user IR bits will be overwritten.
- By using the all-1 opcode that is mandated by the IEEE 1149.1 standard.

Arguments

- `count : integer ;`

A property to request a specific number of data-out ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d).

- `port_naming : port_naming ;`

A property that specifies the naming of the data-out ports. The *port_naming* value has two distinct formats: a single string using an index symbol, or a list of port names with explicit sizes.

When using the indexed format, you specify “base_name[%d]” when you want to use a bus and you specify “base_name%<integer>d” when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, “Dout%2d” will create Dout00, Dout01, Dout02, and so on. “Dout%d” and “Dout%1d” mean the same thing. The default is “user_ir_bits[%d]”.

When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly 7 data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in [DataOutPort Connection Multiplexing](#) of the Tdr/[DataOutPorts](#).

- `multiplexing : on | off | auto ;`

A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See this illustrated in [DataOutPort Connection Multiplexing](#) of the Tdr/DataOutPorts wrapper.

- `connection(range) : pin_name ;`

A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the “[process_dft_specification -no_insertion](#)” option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of four data-out ports with connections to pins on a design instance.

```
DftSpecification(MyCore,rtl) {
    IjtagNetwork {
        HostScanInterface(purple) {
            Tap(orange) {
                DataOutPorts {
                    connection(0) : u2/top_green/en ;
                    connection(3:1) : u2/top_green	mode[2:0] ;
                }
            }
        }
    }
}
```

Related Topics

[Tdr/DataOutPorts](#)

Tap/DeviceIDRegister

A wrapper that specifies that the TAP controller is to be equipped with a device id register.

Usage

```
DftSpecification(module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                DeviceIDRegister {
                    Interface {
                        version_code      : port_name ; // def: version_code[%d]
                        part_number_code : port_name ; // def: part_number_code[%d]
                        manufacturer_id_code: port_name ;
                                         // def: manufacturer_id_code[%d]
                    }
                    capture_source      : internal | external ;
                    instruction_codes : auto | binary, ... ;
                    version_code       : binary ;      // default: 4'h0
                    manufacturer_id_code: binary ;    // default: 11'h0
                    part_number_code   : binary ;    // default: 16'h0
                }
            }
        }
    }
}
```

Description

A wrapper that specifies that the TAP controller is to be equipped with a device id register.

Properties are used to specify the version id, manufacturer id, and the part number codes. When the property `capture_source` is set to external, the version and part_number codes are captured from ports on the TAP controller. The port names are user-specifiable using the `Interface` wrapper. The specified version and part_number codes are used as tie values on the version and part_number code ports when instantiating the TAP controller inside the design.

The tool automatically assigns the instruction codes when you specify `auto` for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Arguments

- `Interface/version_code : port_name ;`

This property defines the name of the port labelled “version_code” in [Figure 10-19](#). The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “`version_code[%d]`”.

- `Interface/part_number_code : port_name ;`

This property defines the name of the port labelled “part_number_code” in [Figure 10-19](#). The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “`part_number_code[%d]`”.

- Interface/manufacturer_id_code : *port_name* ;

This property defines the name of the port labelled “manufacturer_id_code” in [Figure 10-19](#). The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “manufacturer_id_code[%d]”.

- capture_source : internal | external ;

A property that defines where the capture value of the part number, version and manufacturer_id codes comes from. When the property is set to external, the manufacturer_id_code, version_code and part_number_code properties are captured from ports on the TAP controller. The manufacturer_id_code, version_code and part_number_code properties are used as tie values on the manufacturer_id, version and part_number ports when instantiating the TAP controller inside the design. When the property is set to internal, no ports are created and the capture value is described in the RTL description of the TAP. If you set this property to internal, you will not be able to have version and part_number codes specified in the BondingConfigurations wrapper of the BoundaryScan wrapper. The default value is external.

- instruction_codes : auto | *binary*, ... ;

A property used to define one or many opcodes for the DEVICE_ID instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- version_code : *binary* ;

A property that specifies a version id code using a 4-bit binary or hexadecimal number. The default value is “4'h0”.

- manufacturer_id_code : *binary* ;

A property that specifies a manufacturer id code using an 11-bit binary or hexadecimal number. The default value is “11'h0”.

- part_number_code : *binary* ;

A property that specifies a part number code using a 16-bit binary or hexadecimal number. The default value is “16'h0”.

Examples

This example specifies that the TAP controller is to have a device id register. The version and part number codes are encoded as tie values on ports of the TAP controller. This will make it possible to have version and *part_number* codes specified in the BondingConfigurations wrapper of the BoundaryScan wrapper.

```
DftSpecification (module_name, id) {
    IJtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                DeviceIDRegister {
                    version_code      : 4'h0 ;
                    manufacturer_id_code : 11'hA3 ;
                    part_number_code   : 16'h0105 ;
                }
            }
        }
    }
}
```

Tap/HostIjtag

A repeatable wrapper that specifies the presence of one or many IJTAG host ports on the TAP controller.

Usage

```
DftSpecification(module_name, id) {
    IjtagNetwork {
        HostScanInterface(id) {
            Tap(id) {
                HostIjtag(id) {
                    instruction_codes : auto | binary, ... ;
                    Interface {
                        select          : port_name ; // default: host_%s_to_sel
                        scan_in         : port_name ; // default: host_%s_from_so
                    }
                    Sib(id) {
                    }
                    Tdr(id) {
                    }
                    ScanMux(id) {
                    }
                    DesignInstance(instance_name) {
                    }
                }
            }
        }
    }
}
```

Description

A repeatable wrapper that specifies the presence of one or many IJTAG host ports on the TAP controller.

The optional properties allow you to specify a specific opcode to access each IJTAG host port, and to name the ports. You can then specify any other IJTAG node types inside it.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Arguments

- *id*
A string that uniquely identifies the HostIjtag wrapper. The string contains only letters, numbers, and underscores.
- **instruction_codes** : auto | binary, ... ;
A property that defines one or many opcodes for the instruction that selects the IJTAG host port. All codes must differ from the other codes specified in the other wrappers such as inside the [Tap/HostBScan](#) wrapper by at least one non-X bit and the width must be the

same. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- Interface/select : *port_name* ;

This property defines the name of the port labelled “hostX_select” in [Figure 10-19](#). The specified value may include the %s symbol which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_sel”.

- Interface/scan_in : *port_name* ;

This property defines the name of the port labelled “hostX_from_so” in [Figure 10-19](#). The specified value may include the %s symbol which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_from_so”.

Examples

This example defines two IJTAG host scan interfaces. The default opcodes and port naming are used.

```
DftSpecification(my_chip,rtl) {
    IjtagNetwork {
        HostScanInterface(tap) {
            Tap(main) {
                HostIjtag(mission) {
                }
                HostIjtag(debug) {
                }
            }
        }
    }
}
```

InSystemTest

Specifies the generation and optional insertion of an IST controller in the design using the DftSpecification wrapper.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            host_interface      : name;           // mutually exclusive
                                                // with DesignInstance
            DesignInstance (instance_name) { // mutually exclusive
                client_interface: name;           // with host_interface
            }
            protocol          : direct_memory_access | cpu_interface;
            data_width        : int;             // required
            parent_instance   : name;           // default: ""
            leaf_instance_name : name;
            async_reset_all_registers : on | off;
            DirectMemoryAccessOptions {
            }
            Interface {
            }
            Connections {
            }
        }
    }
}
```

Description

InSystemTest is an optional wrapper within the DFTSpecification that you use to generate and optionally insert an IST controller in the design. You can run in-system tests during power up of the device (Power-On Self Test or POST) or run on demand during system operation.

The IST controller is a unique, simple alphanumeric *id*. The *id* should start with an alphanumeric character and can contain no special characters except for underscores (_) after the first character. For example, the following controller ids are valid:

```
Controller(1)
Controller(c1)
Controller(controller_1)
```

The Controller wrapper is a repeatable wrapper. You can specify more than one IST controller.

Specify the IST controller using the host_interface property or the DesignInstance wrapper. Use the host_interface property to specify the scan interface that connects the IST controller when the IJTAG network is inserted at the same time as the IST controller. The *name* is the HostScanInterface() wrapper name inside the IjtagNetwork specification.

Use the DesignInstance wrapper to specify the scan interface connected to the controller when the IJTAG network was inserted prior to the IST controller. The *instance_name* is an instance

name of a module with a matching ICL description. Specifying an IST controller without either a pre-existing or co-generated intercept is not allowed.

For details about Tessent MissionMode, refer to the [Tessent MissionMode User's Manual](#).

Arguments

- host_interface : *name* ;

A required property that specifies the scan interface that will be connected to the controller when the IJTAG network is inserted at the same time as the controller. This property points to the HostScanInterface() wrapper inside the IjtagNetwork Specification. This is the interface at which the controller is connected and the in-system test applied. This property cannot be specified with the DesignInstance() wrapper.

- DesignInstance(*instance_name*) {
 client_interface : *name* ;
}

A required wrapper that specifies a design instance with a scan interface that is to be connected to the controller when the IJTAG network is inserted prior to the controller generation. The id of the wrapper is an *instance_name* of a module with matching ICL description or “.” for the current design.

If there are multiple client scan interfaces in the specified instance, point to a single client using the client_interface property.

This wrapper cannot be specified with the host_interface property.

- protocol : direct_memory_access | cpu_interface ;

An optional property that specifies whether a direct memory access or CPU-based access controller is generated. A direct memory access controller is generated if this property is not specified.

- data_width : *int* ;

An required property that specifies the word size of the memory for the direct memory access controller or system data bus width for a CPU-based access controller.

- parent_instance : *name* ;

An optional property that specifies the parent instance *name* of the IST controller. The default is an empty string, “”. The controller is inserted at the top level of the design when this property is not specified.

- leaf_instance_name : *name* ;

An optional property that specifies the instance *name* of the IST controller. The default for *name* is “*designname_passID_tessent_in_system_test_ctrlID*” and cannot be changed.

- async_reset_all_registers : on | off ;

An optional property that specifies whether all registers will be reset using an asynchronous reset input port. By default (off), only a subset of registers that are functionally required to

be reset (such as the FSM state register) will have asynchronous reset. Set this property to on to ensure all registers in the IST controller have an asynchronous reset; this may be required to conform to certain RTL coding guidelines that require all registers to have asynchronous control.

Examples

In the following example, the controller attached to a TAP controller is generated in the same session as the IST controller:

```
DftSpecification(piccpu,rtl) {
    IJtagNetwork {
        HostScanInterface(dft) {
            Tap(t1) {
                HostIjttag(i1) {
                    Sib(sri) {
                        ...
                    }
                    Sib(sti) {
                        ...
                    }
                }
            }
        }
    }
    MemoryBist {
        ijtag_host_node : Sib(mbist);
        ...
    }
    OCC {
        ijtag_host_interface : Sib(occ);
        ...
    }
    InSystemTest {
        Controller(c0) {
            protocol : cpu_interface;
            host_interface : HostScanInterface(dft);
            Connections {
                reset : istb/reset;
                CpuInterface {
                    clock : istb/clock;
                    enable : istb/enable;
                    write_en : istb/write_en;
                    data_in : istb/data_out;
                    data_out : istb/data_in;
                }
            }
        }
    }
}
```

The following dofile snippet shows that you are inserting three IST controllers. These controllers drive the IJTAG scan interface of the design instances m8051_A, m8051_B and piccpu. These are either sub-blocks or physical blocks on which the IJTAG network was previously inserted.

```
...
}

InSystemTest {
    Controller(m8051_A) {
        protocol: cpu_interface;
        DesignInstance(m8051_A) {
            }
        data_width: 8;
        Connections {
            reset: istb/reset;
            CpuInterface {
                clock: istb/clock;
                enable: istb/en1;
                write_en: istb/write_en;
                data_in: istb/data_out[7:0];
                data_out: istb/data_in1;
            }
        }
    }

    Controller(m8051_B) {
        protocol: cpu_interface;
        DesignInstance(m8051_B) {
            client_interface: C0;
        }
        data_width: 10;
        Connections {
            reset: istb/reset;
            CpuInterface {
                clock: istb/clock;
                enable: istb/en2;
                write_en: istb/write_en;
                data_in: istb/data_out[9:0];
                data_out: istb/data_in2;
            }
        }
    }

    Controller(piccpu) {
        protocol: cpu_interface;
        DesignInstance(piccpu) {
            }
        data_width: 12;
        Connections {
            reset: istb/reset;
            CpuInterface {
                clock: istb/clock;
                enable: istb/en3;
                write_en: istb/write_en;
                data_in: istb/data_out;
                data_out: istb/data_in3;
            }
        }
    }
}
```

DirectMemoryAccessOptions

An optional wrapper that specifies properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            DirectMemoryAccessOptions {
                begin_address      : int; // default: 0
                end_address        : int; // default:
                                         // max addressable memory
                address_width     : int; // required
                max_wait_cycles   : int; // default: 1 million (20 bits)
                max_test_program_count : int; // default: 1
            }
        }
    }
}
```

Description

An optional wrapper that groups properties related to the direct memory access controller.

The begin_address and end_address arguments specify the memory space that is utilized for in-system test when sharing with a functional memory. The controller address counter is set to begin_address when the controller is reset. The tool generates an error message during pattern generation if the entire program does not fit in memory between the begin_address and end_address.

Arguments

- begin_address : *int* ;
An optional property used to specify the begin address for the memory block. The default is 0.
- end_address: *int* ;
An optional property used to specify the end address for the memory block. The default is the max addressable memory size.
- address_width: *int* ;
A required property used to specify the design memory width.
- max_wait_cycles: *int* ;
An optional property used to specify the max wait cycles. The default is 1 million cycles, which corresponds to a 20 bit wait register.
- max_test_program_count: *int* ;
An optional property used to specify the max test program count. The default is 1.

Interface

An optional wrapper that specifies the interface port names for the IST controller.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Interface {
                reset : port_name;      // default: reset
                reset_polarity: active_high | active_low;
                test_active : port_name; // default: test_active
                CpuInterface {
                }
                DirectMemoryAccess {
                }
                TapScanInterface {
                }
                IjtagScanInterface {
                }
            }
        }
    }
}
```

Description

An optional wrapper that specifies the interface port names for the IST controller.

The TapScanInterface tck, trst, tms, tdi, tdo, and tdo_en ports specify the client side of the controller connected to the design source pins (that is, the pads). Ports to_tck, to_trst, to_tms, to_tdi, from_tdo, and from_tdo_en specify the host side of the controller connected to the instrument interface, such as TAP controller pins.

Arguments

- **reset : *port_name* ;**
An optional property that defines the asynchronous reset input port of the IST controller.
- **reset_polarity : active_high | active_low ;**
An optional property that defines the polarity of the reset input port. A reset polarity of active_high applies the reset. A reset polarity of active_low does not apply the reset. The default is active_low.
- **test_active : *port_name* ;**
An optional property that defines the test active output port name. The default *port_name* is test_active. This property is typically used to make connections when test mode is running. For example, use the test_active property to configure the memory to be accessible by the IST controller during test.

CpuInterface

An optional wrapper defined within the Interface wrapper that specifies interface port names for the CPU-based access IST controller.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Interface {
                CpuInterface {
                    clock      : port_name; // default: cpu_interface_clock
                    data_in    : port_name; // default: data_from_cpu[%d]
                    data_out   : port_name; // default: data_to_cpu[%d]
                    write_en   : port_name; // default: write_en_from_cpu
                    enable     : port_name; // default: cpu_interface_en
                }
            }
        }
    }
}
```

Description

The CpuInterface wrapper is an optional wrapper that defines interface port names for the CPU-based access controller. The tool generates the CPU-based access controller based on a DftSpecification. During IJTAG pattern generation, the tool generates a Verilog testbench that uses Verilog tasks to mimic CPU operation. This testbench directly forces and observes the internal design pins and cannot be used as a tester program. All control signals are asserted at the internal pins on the controller boundary.

Arguments

- **clock : *port_name* ;**
An optional property that specifies the name of the clock input for the controller. The default *port_name* is `cpu_interface_clock`. The CPU's native clock is not required to be synchronous with this clock. This clock will drive the connected IJTAG network in test mode.
- **data_in : *port_name* ;**
An optional property that specifies the name of the parallel input bus port. The default *port_name* is “`data_from_cpu[%d]`”, where `%d` is a literal string that is expanded to the range of the vector. This bus contains IST controller instruction opcode and data to write to the IJTAG network through the TAP controller. The two most significant bits represent an opcode for the controller and the remaining bits are the write data that will be shifted into the network.
- **data_out : *port_name* ;**
An optional property that specifies the name of the parallel output bus port. The default *port_name* is “`data_to_cpu[%d]`”, where `%d` is a literal string that is expanded to the range

of the vector. This bus contains the status of the controller read operation and the data read from the IJTAG network. The two most significant bits represent a status code and the remaining bits are the read data shifted out from the network.

- `write_en : port_name ;`

An optional property that specifies the name of the IST controller write enable input port. The default *port_name* is `write_en_from_cpu`. This input signal indicates valid data is present in the DataIn bus and is to be asserted by the CPU after writing to DataIn.

- `enable : port_name ;`

An optional property that specifies the name of the IST controller enable port. The default *port_name* is `cpu_interface_en`. This port is the input signal used to enable InSystemTest mode. This signal should be asserted and stay asserted during the entire test. This signal is also used outside the controller to provide multiplexed access to the IJTAG network inputs from the controller.

DirectMemoryAccess

An optional wrapper defined within the Interface wrapper that groups properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Interface {
                DirectMemoryAccess {
                    clock : port_name; // default: dma_clock
                    test_program_index: port_name; // default:
                                                // dma_test_program[%d]
                    test_program_done : port_name; // default:
                                                // dma_test_program_done
                    fail_flag : port_name; // default: dma_fail_flag
                    mem_address : port_name; // default: dma_address[%d]
                    mem_data : port_name; // default: dma_data[%d]
                    enable : port_name; // default: dma_en
                }
            }
        }
    }
}
```

Description

The DirectMemoryAccess wrapper is an optional wrapper defined within the Interface wrapper that groups properties related to the direct memory access controller. This wrapper is applied when using the direct memory access protocol.

Arguments

- **clock : *port_name* ;**
An optional property that specifies the name of the controller or direct memory access controller clock input port. The default *port_name* is *dma_clock*.
- **test_program_index : *port_name* ;**
An optional property that specifies the name of the test program index input port. The default *port_name* is *dma_test_program[%d]*, where %d is a literal string that is expanded to the range of the vector during execution. Use this property only when multiple test programs exist.
- **test_program_done : *port_name* ;**
An optional property that specifies the name of the test program done output port. The default *port_name* is *dma_test_program_done*.

- `fail_flag : port_name ;`

An optional property that specifies the name of the fail flag output port. The default *port_name* is `dma_fail_flag`. Use this port to determine test status. The signal indicates when mismatches between read and expect data has occurred. This signal is cleared when the FSM goes to the Idle state.

- `mem_address : port_name ;`

An optional property that specifies the name of the memory address output port. The default *port_name* is “`dma_address[%d]`”, where `%d` is a literal string that is expanded to the range of the vector. This bus provides the address for the next memory read. When memory is shared with functional logic, ensure the controller has exclusive access to the memory address bus.

- `mem_data : port_name ;`

An optional property that specifies the name of the memory data input port. The default *port_name* is “`dma_data[%d]`”, where `%d` is a literal string that is expanded to the range of the vector during processing. Use this input bus to provide the data corresponding to the current address.

- `enable : port_name;`

An optional property that specifies the name of the enable input port. The default *port_name* is `dma_en`. Use this port to start the in-system test. The enable must stay active for the entire duration of the test. This signal can be de-asserted at anytime to stop a running test.

TapScanInterface

An optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses a TAP controller to execute tests.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Interface {
                TapScanInterface {
                    tck      : port_name; // default: ijttag_tck
                    trst     : port_name; // default: trst
                    tms      : port_name; // default: tms
                    tdi      : port_name; // default: tdi
                    tdo      : port_name; // default: tdo
                    tdo_en   : port_name; // default: tdo_en
                    to_tck   : port_name; // default: to_ijtag_tck
                    to_trst  : port_name; // default: to_trst
                    to_tms   : port_name; // default: to_tma
                    to_tdi   : port_name; // default: to_tdi
                    from_tdo : port_name; // default: from_tdo
                    from_tdo_en: port_name; // default: from_tdo_en
                }
            }
        }
    }
}
```

Description

The TapScanInterface wrapper defined within the Interface wrapper is an optional wrapper that specifies the interface port names when the IST controller uses a TAP controller to execute tests.

The top-level TAP signals are tck, trst, tms, tdi, and tdo. These signals are brought into the controller for multiplexing with the controller generated signals. The output signals from the controller are to_tck, to_trst, to_tms, to_tdi, and from_tdo.

Arguments

- `tck : port_name ;`
An optional property that specifies the name of the test clock input port. The default *port_name* is ijttag_tck.
- `trst : port_name ;`
An optional property that specifies the name of the test reset input port. The default *port_name* is trst.
- `tms : port_name ;`
An optional property that specifies the name of the tms input port. The default *port_name* is tms.

- `tdi : port_name ;`
An optional property that specifies the name of the tdi input port. The default *port_name* is `tdi`.
- `tdo : port_name ;`
An optional property that specifies the name of the tdo output port. The default *port_name* is `tdo`.
- `tdo_en : port_name ;`
An optional property that specifies the name of the tdo enable input port. The default *port_name* is `tdo_en`.
- `to_tck : port_name ;`
An optional property that specifies the name of the output clock port. The default *port_name* is `to_ijtag_tck`. This property handles the output signal from the IST controller. When the controller is active the clock input of the controller drives this port, otherwise the tck input drives this port.
- `to_trst : port_name ;`
An optional property that specifies the name of the output test reset port. The default *port_name* is `to_trst`. This property handles the output signal from the IST controller that is held inactive when the controller is running, otherwise it is driven by the top-level trst port.
- `to_tms : port_name ;`
An optional property that specifies the name of the output tms port. The default *port_name* is `to_tms`. This property handles the output signal from the IST controller that is multiplexed with the top-level tms port at the input of the TAP controller.
- `to_tdi : port_name ;`
An optional property that specifies the name of the output tdi port. The default *port_name* is `to_tdi`. This property handles the output signal from the IST controller that is multiplexed with the top-level tdi port at the source of the TAP controller `tdi` input.
- `from_tdo : port_name ;`
An optional property that specifies the name of the from tdo output port. The default *port_name* is `from_tdo`. This property handles the input signal to the IST controller that comes from the internal tdo signal (before the tristate buffer) of the TAP controller.
- `from_tdo_en : port_name ;`
An optional property that specifies the name of the output tdo enable port. The default *port_name* is `from_tdo_en`.

IjtagScanInterface

An optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses an IJTAG scan interface to execute tests.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Interface {
                IjtagScanInterface {
                    tck      : port_name; // default: ijtag_tck
                    reset   : port_name; // default: ijtag_reset
                    select   : port_name; // default: ijtag_sel
                    capture_en : port_name; // default: ijtag_ce
                    shift_en  : port_name; // default: ijtag_se
                    update_en : port_name; // default: ijtag_ue
                    scan_in   : port_name; // default: ijtag_si
                    scan_out   : port_name; // default: ijtag_so

                    to_tck      : port_name; // default: to_ijtag_tck
                    to_reset   : port_name; // default: to_ijtag_reset
                    to_select   : port_name; // default: to_ijtag_sel
                    to_capture_en: port_name; // default: to_ijtag_ce
                    to_shift_en : port_name; // default: to_ijtag_se
                    to_update_en : port_name; // default: to_ijtag_ue
                    to_scan_in   : port_name; // default: to_ijtag_si
                    from_scan_out: port_name; // default: from_ijtag_so
                }
            }
        }
    }
}
```

Description

The IjtagScanInterface wrapper is an optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses an IJTAG scan interface.

Arguments

- **tck : *port_name*;**
An optional property that specifies the name of the test clock port. The default *port_name* is ijtag_tck.
- **reset : *port_name*;**
An optional property that specifies the name of the reset port. The default *port_name* is ijtag_reset.
- **select : *port_name*;**
An optional property that specifies the name of the select port. The default *port_name* is ijtag_sel.

- `capture_en : port_name;`
An optional property that specifies the name of the enable capture port. The default *port_name* is `ijtag_ce`.
- `shift_en : port_name;`
An optional property that specifies the name of the enable shift port. The default *port_name* is `ijtag_se`.
- `update_en : port_name;`
An optional property that specifies the name of the enable update port. The default *port_name* is `ijtag_ue`.
- `scan_in : port_name;`
An optional property that specifies the name of the scan in port. The default *port_name* is `ijtag_si`.
- `scan_out : port_name;`
An optional property that specifies the name of the scan out port. The default *port_name* is `ijtag_so`.
- `to_tck : port_name;`
An optional property that specifies the name of the output clock port. The default *port_name* is `to_ijtag_tck`.
- `to_reset : port_name;`
An optional property that specifies the name of the output reset port. The default *port_name* is `to_ijtag_reset`.
- `to_select : port_name;`
An optional property that specifies the name of the select output port. The default *port_name* is `to_ijtag_sel`.
- `to_capture_en : port_name;`
An optional property that specifies the name of the output capture enable port. The default *port_name* is `to_ijtag_ce`.
- `to_shift_en : port_name;`
An optional property that specifies the name of the output shift enable port. The default *port_name* is `to_ijtag_se`.
- `to_update_en : port_name;`
An optional property that specifies the name of the output enable update port. The default *port_name* is `to_ijtag_ue`.
- `to_scan_in : port_name;`
An optional property that specifies the name of the output scan in port. The default *port_name* is `to_ijtag_si`.

- `from_scan_out : port_name;`

An optional property that specifies the name of the from scan output port. The default *port_name* is `from_ijtag_so`.

Connections

An optional wrapper used to define the design node used to connect the corresponding IST controller pin.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Connections {
                reset           : port_pin_name; // required
                test_active     : port_pin_name; // default: "" (no connection)
                CpuInterface {
                    // required
                }
                DirectMemoryAccess {
                    // required
                }
            }
        }
    }
}
```

Description

The Connections wrapper is an optional wrapper used to define the design node used to connect the corresponding IST controller pin. Connections do not exist for the TapScanInterface intercept pins. The intercept pins are connected to the location defined by host_interface or DesignInstance() arguments.

When design nodes for the Connections wrapper do not exist, or you do not want to make connections, specify “0” for the input *port_pin_name* and empty string, “”, for the output *port_pin_name*. This includes the reset and test_active properties, as well as the properties defined within the CpuInterface and DirectMemoryAccess wrappers.

Pin names should be existing pin names from the design. Port names can be any name, but must be the correct direction.

Arguments

- *reset* : *port_pin_name*;
A required property that specifies the reset input port or pin name. This property specifies the asynchronous input, for example a design’s reset or poweron_reset.
- *test_active* : *port_pin_name*;
An optional property that specifies the test active output port or pin name. The default is an empty string, “”, if there is no connection.

CpuInterface

A required wrapper defined within the Connections wrapper that specifies the required input and output port or pin names used by the CPU interface.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Connections {
                CpuInterface {
                    clock : port_pin_name; // required input
                    data_in : port_pin_name; // required input
                    data_out: port_pin_name; // required output
                    write_en: port_pin_name; // required input
                    enable : port_pin_name; // required input
                }
            }
        }
    }
}
```

Description

The CpuInterface wrapper is a required wrapper defined within the Connections wrapper that specifies the required input and output port or pin names used by the CPU interface.

When design nodes for the Connections wrapper do not exist or you do not want to make connections, specify “0” for the input *port_pin_name* and empty string, “”, for the output *port_pin_name*.

Arguments

- *clock* : *port_pin_name* ;
A required property that specifies the clock input port or pin name. The CPU’s native clock is not required to be synchronous with this clock. This clock will drive the connected IJTAG network in test mode.
- *data_in* : *port_pin_name* ;
A required property that specifies the name of the parallel input bus port or pin name. This bus contains IST controller instruction opcode and data to write to the IJTAG network through the TAP controller. The two most significant bits represent an opcode for the controller and the remaining bits are the write data that will be shifted into the network.
- *data_out* : *port_pin_name* ;
A required property that specifies the name of the parallel output bus port or pin name. This bus contains the status of the controller read operation and the data read from the IJTAG network. The two most significant bits represent a status code and the remaining bits are the read data shifted out from the network.

- `write_en : port_pin_name ;`

A required property that specifies the write enable input port or pin name. This input signal indicates valid data is present in the DataIn bus and is to be asserted by the CPU after writing to DataIn.

- `enable : port_pin_name ;`

A required property that specifies the name of the IST controller enable input port or pin name. This port is the input signal used to enable In-System Test mode. This signal should be asserted and stay asserted during the entire test. This signal is also used outside the IST controller to provide multiplexed access to the IJTAG network inputs from the controller.

DirectMemoryAccess

A required wrapper defined within the Connections wrapper that groups properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
    InSystemTest {
        Controller(id) {
            Connections {
                DirectMemoryAccess {
                    clock : port_pin_name; // required
                    test_program_index: port_pin_name; // required
                    test_program_done : port_pin_name; // required
                    fail_flag : port_pin_name; // required
                    mem_address : port_pin_name; // required
                    mem_data : port_pin_name; // required
                    enable : port_pin_name; // required
                }
            }
        }
    }
}
```

Description

The DirectMemoryAccess wrapper is a required wrapper defined within the Connections wrapper that groups properties related to the direct memory access controller.

When design nodes for the Connections wrapper do not exist or you do not want to make connections, specify “0” for the input *port_pin_name* and empty string, “”, for the output *port_pin_name*.

Arguments

- *clock* : *port_pin_name* ;

A required property that specifies the input clock port or pin name for the IST controller. The CPU’s native clock is not required to be synchronous with this clock. This clock will drive the connected IJTAG network in test mode.

- *test_program_index* : *port_pin_name* ;

A required property that specifies the name of the test program index input port. Use this property only when multiple test programs exist.

- *test_program_done* : *port_pin_name* ;

A required property that specifies the test program done output port or pin name.

- *fail_flag* : *port_pin_name* ;

A required property that specifies the fail flag output port or pin name. Use this port to determine test status. The signal indicates when mismatches between read and expect data have occurred. This signal is cleared when the FSM goes to the Idle state.

- mem_address : *port_pin_name* ;
A required property that specifies the memory address output port or pin name. This bus provides the address for the next memory read. When memory is shared with functional logic, ensure the controller has exclusive access to the memory address bus.
- mem_data : *port_pin_name* ;
A required property that specifies the memory data input port or pin name. This bus provides the data corresponding to the current address.
- enable : *port_pin_name* ;
A required property that specifies the input enable port or pin name. This input enables InSystemTest mode and should be asserted and stay asserted during the entire test. You can also use this signal outside the IST controller to provide multiplexed access to the IJTAG network inputs from the controller.

EmbeddedBoundaryScan

Specifies the creation of a boundary scan segment inside a module.

Usage

```
DftSpecification(module_name,id) {
    EmbeddedBoundaryScan {
        pad_io_ports           : port_name, ... ;
        interface_parent_instance : parent_instance ;
        outputs_per_enable_cell : int ; // default: 16 *DefSpec
        Interface {
            // *DefSpec
        }
        ImplementationOptions { // *DefSpec
        }
        BoundaryScanCellOptions {
        }
        LogicalGroups {
        }
        AuxiliaryInputOutputPorts {
        }
        ACModeOptions {
        }
        EnableGroups {
        }
        InternalBScanCells {
        }
        InternalBscanSegments {
        }
    }
}
```

Description

A wrapper that specifies the creation of a boundary scan segment inside a module. This module can then be reused as a building block when constructing the boundary scan chain at the next level up. This feature is useful in two situations. The first usage is when you have a physical region containing pads and you want to insert the boundary-scan cells into it before laying it out. The second usage is when you want to insert the boundary-scan cells into non-unique sub-modules without having to uniquify them. You can make the sub-module the current design, insert a boundary scan segment into it, and then reuse the module more than once at the next level.

A *.bcd* file containing a BoundaryScan wrapper (see “[Tessent Core Description](#)” on page 3755) is created for the current design. Once loaded in memory, the tool automatically recognizes the existing boundary scan segments in the design and includes them in the boundary scan chain at the current level. The legacy *.lvbscan* file format is also natively supported and translated into the current BoundaryScan wrapper syntax when loaded into Tessent Shell. Refer to the [set_module_matching_options](#) and the “[set_design_sources -format](#)” commands to learn how to have them automatically searched and loaded. Refer to the [read_core_descriptions](#) command to learn how to explicit load them in.

Arguments

- `pad_io_ports : port_name, ... ;`

A property that lists all ports that are to be equipped with a boundary-scan cell. Those ports must all be directly connected to an input and/or output pad cell. The order of the list is used to order the boundary scan segment where the cell associated with the first port is closer to scanin and the cell associated with the last port is closer to scanout. You have to list each port one by one; no wildcards are allowed. However you can use the `%#d[left_bit:right_bit]` syntax to specify a group of numbered ports. A bused port can be specified with the order of the index following the required order of the boundary scan segment, and it does not necessarily have to be the order in which it is declared in the design. For example, the bus `GPIO[7:0]` can be specified as `GPIO[4:7], GPIO[0:3]` to specify that bit 4 is closest to scanin and is followed by bit 5 even if the bus is declared with the larger index on the left.

This property is automatically populated by the [create_dft_specification](#) command with the port list you have specified with the [set_boundary_scan_port_options](#) -pad_io_ports option. The port list is physically ordered if you read in a DEF file using the [read_def](#) command.

- `interface_parent_instance : parent_instance ;`

An optional property that identifies the design module in which the tool will place the boundary scan interface. The default placement will be at the top level if this property is not specified.

- `outputs_per_enable_cell : int ;`

An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. If the enable pin of the output pad cell is tied on or off, the tool is free to assign any output pad cell to any given enable cell in order to meet the outputs_per_enable_cell constraints. If, on the other hand, a group of output pad cells have their enable pin controlled by a single functional net, the tool will insert N enable boundary-scan cells and automatically separate the fanout of the enable signal such that the fanout of each enable cell is smaller than the specified outputs_per_enable_cell value. The splitting of the functional enable fanout is illustrated in the example of the [LogicalGroups](#) wrapper in [Figure 10-25](#) and [Figure 10-26](#).

Examples

The following example defines a boundary scan chain associated with the `GPIO[7:0]` ports inside a module. This module will then be used as a building block for the boundary scan chain at the next level up, either to construct a larger segment in a parent module or as part of the chain for a chip.

```
DftSpecification(my_block,rtl) {
    EmbeddedBoundaryScan {
        pad_io_ports : GPIO[4:7], GPIO[0:3] ;
    }
}
```

Interface

Specifies the naming of all ports created on the current design to provide access to the boundary scan segment.

Usage

```
DftSpecification(module_name, id) {
    EmbeddedBoundaryScan {
        Interface { *DefSpec
            select : port_naming ; // bscan_select
            reset : port_naming ; // bscan_reset
            force_disable : port_naming ; // bscan_force_disable
            select_jtag_input : port_naming ; // bscan_select_jtag_input
            select_jtag_output : port_naming ; // bscan_select_jtag_output
            bscan_clock : port_naming ; // bscan_clock
            capture_en : port_naming ; // bscan_capture_en
            shift_en : port_naming ; // bscan_shift_en
            update_en : port_naming ; // bscan_update_en
            scan_in : port_naming ; // bscan_scan_in
            scan_out : port_naming ; // bscan_scan_out
            auxiliary_output : port_naming ; // bscan_%s_aux_out
            auxiliary_output_enable : port_naming ; // bscan_%s_aux_out_en
            auxiliary_input : port_naming ; // bscan_%s_aux_in
            auxiliary_input_enable : port_naming ; // bscan_%s_aux_in_en
            ac_init_clock0 : port_naming ; // bscan_ac_init_clk0
            ac_init_clock1 : port_naming ; // bscan_ac_init_clk1
            ac_signal : port_naming ; // bscan_ac_signal
            ac_mode_en : port_naming ; // bscan_ac_mode_en
        }
    }
}
```

Description

A wrapper that specifies the naming of all ports created on the current design to provide access to the boundary scan segment.

Arguments

- `select : port_naming ;`

An optional property that defines the name of the select signal. The select signal is used to enable or disable the embedded boundary scan interface of the core. When not specified, the port is named “bscan_select”. Setting this value in the [DefaultsSpecification](#) implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `reset : port_naming ;`

An attribute used to provide the name for the port carrying the reset function. When not specified, the port is named “bscan_reset”. The reset function is active low.

- **force_disable : *port_naming* ;**
An attribute used to provide the name for the port carrying the force_disable function. When not specified, the port is named “bscan_force_disable”.
- **select_jtag_input : *port_naming* ;**
An attribute used to provide the name for the port carrying the select_jtag_input function. When not specified, the port is named “bscan_select_jtag_input”.
- **select_jtag_output : *port_naming* ;**
An attribute used to provide the name for the port carrying the select_jtag_output function. When not specified, the port is named “bscan_select_jtag_output”.
- **bscan_clock : *port_naming* ;**
An attribute used to provide the name for the port carrying the bscan_clock function. When not specified, the port is named “bscan_clock”.
- **capture_en : *port_naming* ;**
An attribute used to provide the name for the port carrying the capture_en function. When not specified, the port is named “bscan_capture_en”.
- **shift_en : *port_naming* ;**
An attribute used to provide the name for the port carrying the shift_en function. When not specified, the port is named “bscan_shift_en”.
- **update_en : *port_naming* ;**
An attribute used to provide the name for the port carrying the update_en function. When not specified, the port is named “bscan_update_en”.
- **scan_in : *port_naming* ;**
An attribute used to provide the name for the port carrying the scan_in function. When not specified, the port is named “bscan_scan_in”.
- **scan_out : *port_naming* ;**
An attribute used to provide the name for the port carrying the scan_out function. When not specified, the port is named “bscan_scan_out”.
- **auxiliary_output : *port_naming* ;**
An attribute used to provide the name for the port carrying the auxiliary_output function. When not specified, the port is named “bscan_%s_aux_out”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output port associated with port GPIO[4] is “bscan_GPIO_4_aux_out”.
- **auxiliary_output_enable : *port_naming* ;**
An attribute used to provide the name for the port carrying the auxiliary_output_enable function. When not specified, the port is named “bscan_%s_aux_out_en”. The presence of

the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output enable port associated with port GPIO[4] is “bscan_GPIO_4_aux_out_en”.

- auxiliary_input : *port_naming*, ... ;

An attribute used to provide the name for the port carrying the auxiliary_input function. When not specified, the port is named “bscan_%s_aux_in”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary input is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input port associated with port GPIO[4] is “bscan_GPIO_4_aux_in”.

- auxiliary_input_enable : *port_naming*, ... ;

An attribute used to provide the name for the port carrying the auxiliary_input_enable function. When not specified, the port is named “bscan_%s_aux_in_en”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input enable port associated with port GPIO[4] is “bscan_GPIO_4_aux_in_en”. This function is rarely used. It is only used when the input pad cell has a built-in JTAG mux or when the input pad buffer has a bscan_select port function.

- ac_init_clock0 : *port_naming* ;

An attribute used to provide the name for the port carrying the ac_init_clock0 function, which has a low “off” state. When not specified, the port is named “bscan_ac_init_clk0”. This port is only created when AC pads are present.

- ac_init_clock1 : *port_naming* ;

An attribute used to provide the name for the port carrying the ac_init_clock1 function, which has a high “off” state. When not specified, the port is named “bscan_ac_init_clk1”. This port is only created when AC pads are present.

- ac_signal : *port_naming* ;

An attribute used to provide the name for the port carrying the ac_signal function. When not specified, the port is named “bscan_ac_signal”. This port is only created when AC pads are present.

- ac_mode_en : *port_naming* ;

An attribute used to provide the name for the port carrying the ac_mode_en function. When not specified, the port is named “bscan_ac_mode_en”. This port is only created when AC pads are present.

Examples

The following example changes the interface port naming by adding the prefix “ts_bscan_” to all created ports. Not all port names will be affected because it will depend on the values

specified in the [ImplementationOptions](#) wrappers and the use of those functions in reused BScanSegment elements.

```
DftSpecification(my_block,rtl) {
    EmbeddedBoundaryScan {
        Interface {
            force_disable : ts_bscan_force_disable ;
            select_jtag_input : ts_bscan_select_jtag_input ;
            select_jtag_output : ts_bscan_select_jtag_output ;
            capture_shift_clock : ts_bscan_cs_clock ;
            capture_shift_clock_inv : ts_bscan_cs_clock_inv ;
            update_clock : ts_bscan_update_clock ;
            capture_en : ts_bscan_capture_en ;
            shift_en : ts_bscan_shift_en ;
            update_en : ts_bscan_update_en ;
            scan_in : ts_bscan_scan_in ;
            scan_out : ts_bscan_scan_out ;
            auxiliary_output : ts_bscan_%s_aux_out ;
            auxiliary_output_enable : ts_bscan_%s_aux_out_en ;
            auxiliary_input : ts_bscan_%s_aux_in ;
            auxiliary_input_enable : ts_bscan_%s_aux_in_en ;
            ac_init_clock0 : ts_bscan_ac_init_clk0 ;
            ac_init_clock1 : ts_bscan_ac_init_clk1 ;
            ac_signal : ts_bscan_ac_signal ;
            ac_mode_en : ts_bscan_ac_mode_en ;
        }
    }
}
```

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        ImplementationOptions {
            clocking           : tck | gated_tck | gated_tck_inv ; // *DefSpec
            update_stage       : flop | latch ;
            scan_path_retiming : flop | latch ;
        }
    }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- **clocking: tck | gated_tck | gated_tck_inv ;**

A property that specifies how the clocking of the boundary-scan cell should be constructed. When specifying tck, the boundary-scan cells are clocked with the free-running TCK clock and the capture and shift is enabled using the capture_en and shift_en functions. This option requires one extra multiplexer per boundary-scan cell as compared to the gated_tck or gated_tck_inv implementations. When specifying gated_tck, a gated low version of TCK is used and the gating lets the TCK go through during the capture and shift enable pulse. When specifying gated_tck_inv, a gated high version of TCK is used and the gating lets the inverted TCK go through during the capture and shift enable pulse. All three clocking specifications behaves identically in IEEE 1149.1 mode. Using TCK versus the other two gated TCK options is a trade-off between area and the number and sizes of clock trees. Note that in all three cases, the boundary-scan cells are always built such that the scanin of each cell is strobed on the rising edge of TCK and the scanout is launched off the negedge of TCK to make sure boundary-scan cells will always work without having to balance TCK to all boundary scan cells which makes the ground bounce issue worst.

The gated_tck_inv option is a legacy clocking mode and is the one that is compatible with the existing LogicVision burst mode architecture. For more information, see the “[BurstMode Logic BIST Architecture](#)” section in the *LV Flow User’s Manual*.

[Table 10-3](#) illustrates the boundary scan cell implementations based on the clocking, update_stage and scan_path_retiming values.

- **update_stage: flop | latch ;**

A property that specifies if the update stage is to be implemented using a latch or a flop.

[Table 10-3](#) illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values.

- scan_path_retiming : flop | latch ;

A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

Table 10-3 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values. Refer to **Table 10-5** to see the implementation of an input and sample_only boundary scan cell using the default implementation options. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

Table 10-3. Illustration of Different Boundary Scan Implementations

- 1) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck, and update_stage and scan_path_retiming are both set to latch.

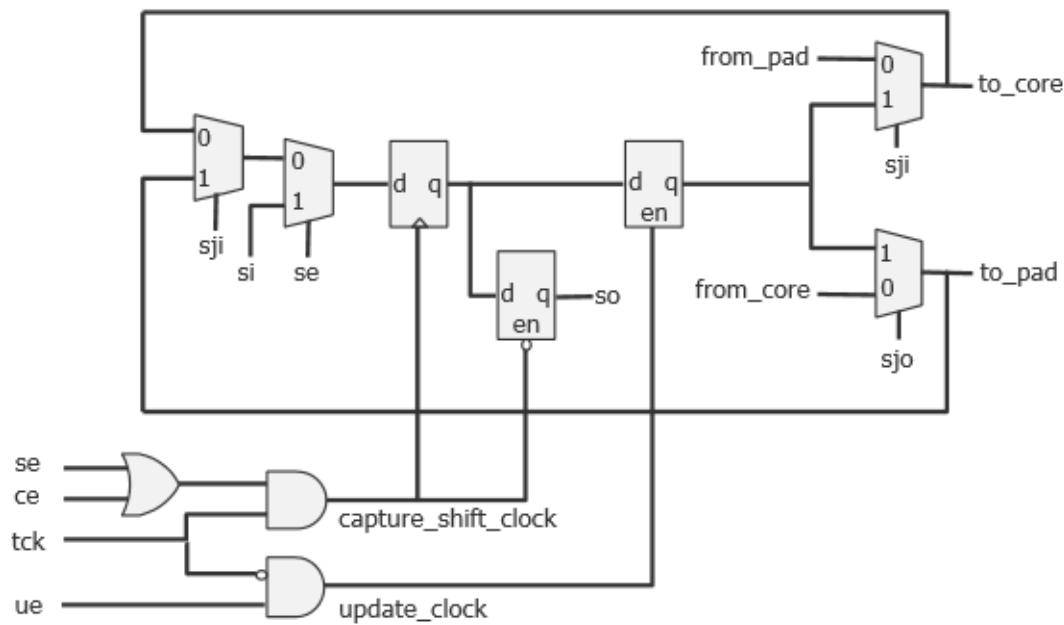


Table 10-3. Illustration of Different Boundary Scan Implementations (cont.)

2) This example shows how the boundary-scan cells are constructed when the clocking is set to gated_tck and update_stage and scan_path_retimng are both set to flop. Compared to the prior example, this shows how changing update_stage and scan_path_retimng from latch to flop simply converts the latch into a flop.

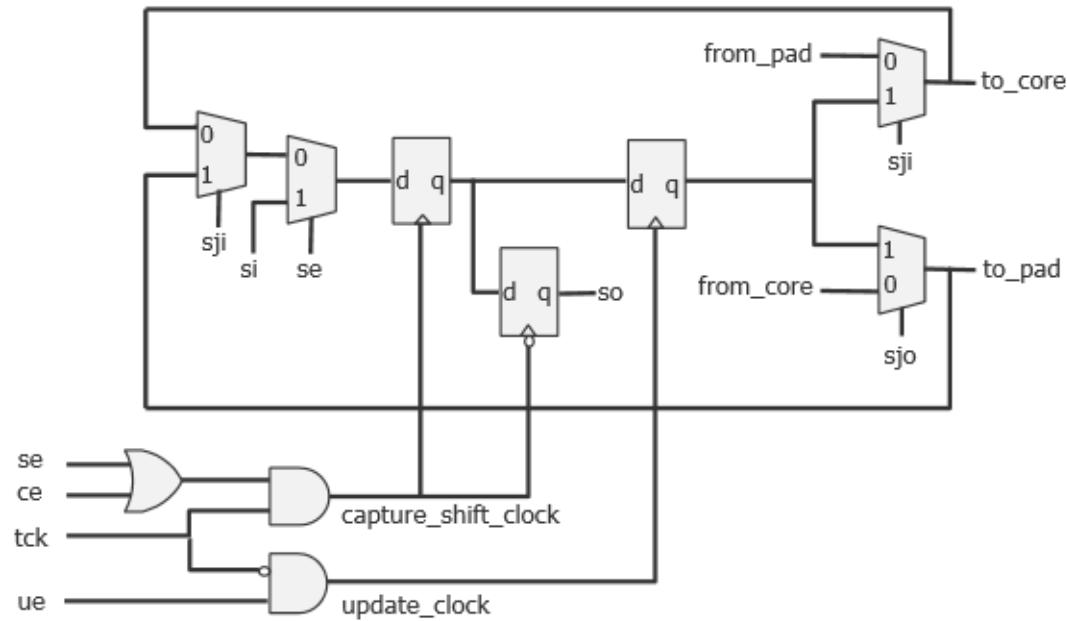
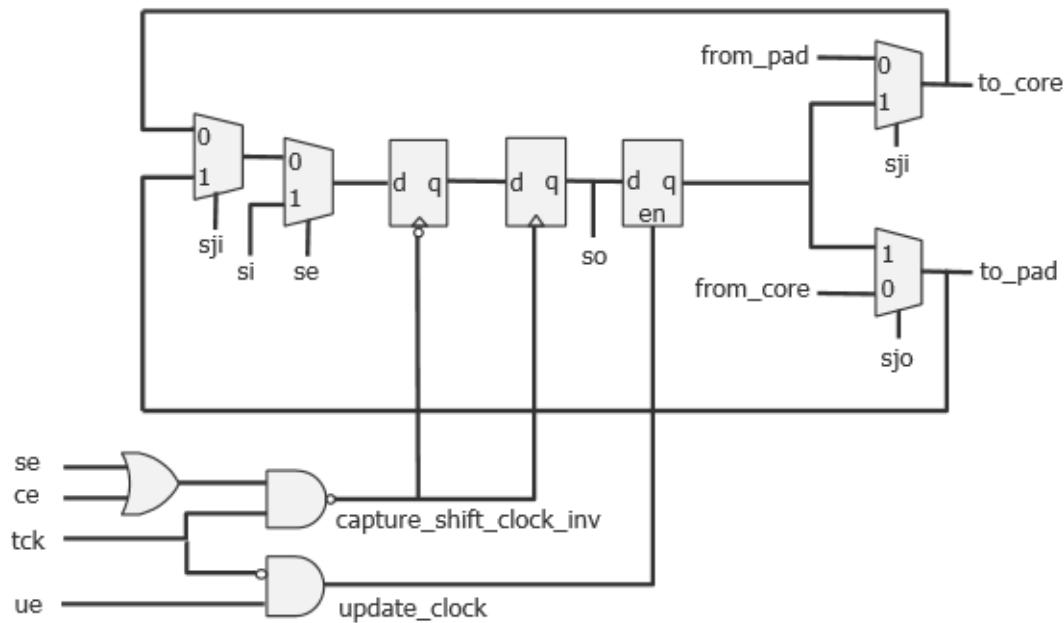
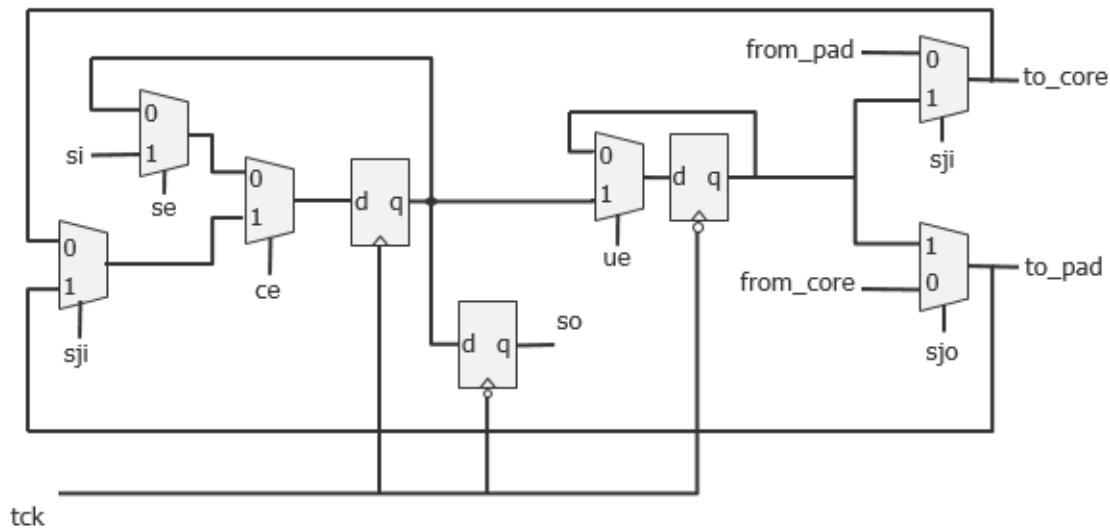


Table 10-3. Illustration of Different Boundary Scan Implementations (cont.)

3) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck_inv, and update_stage and scan_path_retimining are both set to latch. This is a legacy mode that makes the boundary-scan cell compatible with the LogicVision burst mode architecture. Notice how the implementation for gated_tck and gated_tck_inv are almost the same; only the active-low retiming flop is placed before the data flop.



4) This example shows how the boundary-scan cells are constructed when clocking is set to tck, and both update_stage and scan_path_retimining are set to flop. Notice the two extra multiplexers that are needed in each cell.



Examples

The following example sets the clocking property to gated_tck_inv and the update_stage to latch. The scan_path_retiming is unspecified so it defaults to latch.

```
DftSpecification(mychip,rtl) {
    BoundaryScan {
        ImplementationOptions {
            clocking      : gate_tck_inv ;
            update_stage : latch ;
        }
    }
}
```

BoundaryScanCellOptions

Assigns options to boundary scan ports.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        BoundaryScanCellOptions {
            port_name_pattern : option, ... ; // repeatable
        }
    }
}
```

Description

This wrapper is used to assign options to boundary scan ports. This wrapper replaces the Overrides wrapper in the .etassemble file.

Arguments

- *port_pattern_name* : option, ...;

A syntax used to assign one or more option to a set of ports matched by the specified *port_pattern_name*. The *port_pattern_name* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match “ABC00”, “ABC01”, and “ABC02”.

[Table 10-4](#) lists the option names and their meanings. The option can be specified in the BoundaryScanCellOptions wrapper or in the third column of the pin_order_file, as explained in the section.

Table 10-4. BoundaryScanCellOptions list

Option name	Meaning
add_dot6_from_pad_cell	Used to identify AC input and inout ports where the from_pad data signal will be intercepted with an additional boundary scan cell. The boundary scan cells sampling the AC pad Test Receivers will be added as normally done.
analog	Used to identify input, output, and inout ports that are analog. These pins do not have a boundary scan cell and are reported as linkage pins in the BSDL file.

Table 10-4. BoundaryScanCellOptions list (cont.)

Option name	Meaning
compliance_enable0	<p>Used to identify input and inout ports that are active low compliance enable pins. These ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven low for the chip to behave in accordance with the BSDL description. The compliance enable zero pins are automatically asserted low in any pattern that makes use of the TAP controller.</p> <p>This option is automatically set when you use the <code>-enable_low_compliance_enables</code> switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the <code>-enable_low_compliance_enables</code> switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.</p>
compliance_enable1	<p>Used to identify input and inout ports that are active high compliance enable pins. Those ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven high for the chip to behave in accordance with the BSDL description. The compliance enable one ports are automatically asserted high in any pattern that makes use of the TAP controller.</p> <p>This option is automatically set when you use the <code>-enable_high_compliance_enables</code> switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the <code>-enable_high_compliance_enables</code> switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.</p>
clock	<p>Used to identify input, output, and inout ports that are clocks. Specifying this value alters the RTL and the BSDL definition of the generated boundary-scan cells associated with the port.</p> <ul style="list-style-type: none"> • If the port is of type input or inout, the boundary-scan cell will have a BSDL function of “clock” and no JTAG multiplexer is inserted along the functional input path just as if the <code>sample_only</code> option was specified. • If the port is of type output or inout, the boundary-scan cell is modified to capture itself instead of the <code>from_core</code> signal in the output direction just as if the <code>no_capture_core_signal</code> option was specified.

Table 10-4. BoundaryScanCellOptions list (cont.)

Option name	Meaning
dont_touch	Used to identify input, output, and inout ports that are to be fully excluded from boundary scan. It is similar to using the no_bscan_cell option except that for output and inout ports, the output enable pin of the output pad buffer is not intercepted with the force_disable signal such that the port will not go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
ground	Used to identify input and inout ports that are ground pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the ground option may or may not exist in the design. If they do not exist in the design file, the port name pattern is not allowed to include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to ground prior to running the check_design_rules command are automatically identified as ground pins in the created DftSpecification wrapper when you run the create_dft_specification command.
input_only	Used to identify inout ports that are to be treated as input-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file will document the port as input, and the port will only be equipped with an input-type boundary-scan cell.
no_bscan_cell	Used to identify input, output, and inout ports that are not to be equipped with a boundary-scan cell. It is similar to using the dont_touch option except that for output and inout ports, the output enable pin of the output pad buffer is still intercepted with the force_disable signal such that the port will still go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
no_connect	Used to identify input, output, and inout ports that are not connected to package pins. Those ports are treated as dont_touch pins and they are not reported at all in the BSDL file.

Table 10-4. BoundaryScanCellOptions list (cont.)

Option name	Meaning
no_capture_core_signal	Used to identify output and inout ports that are to be equipped with boundary-scan cells that do not capture the functional core signal but instead hold their values during the capture cycle of the INTEST instruction. This option is useful when you are reusing the boundary scan chain as isolation during logictest and you want to avoid the boundary-scan cells from capturing unknowns when the from_core signal is not controllable during logictest. Avoiding unknowns is a requirement if you are using LogicBIST.
output_only	Used to identify inout ports that are to be treated as output-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file will document the port as output, and the port will only be equipped with an output-type boundary-scan cell.
power	Used to identify input and inout ports that are power pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the power option may or may not exist in the design. If they do not exist in the design file, the port name pattern is not allowed to include the asterisk wildcard symbol “*” but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to power prior to running the check_design_rules command are automatically identified as power pins in the created DftSpecification wrapper when you run the create_dft_specification command.
sample_only	Used to identify input and inout ports that are not to have a JTAG multiplexer inserted along the functional input path. The fanout of these ports cannot be intercepted by the boundary-scan cell value during the INTEST instruction. Also, if you reuse the boundary scan as isolation during logictest, these ports will not be isolated. This option is inferred for input or inout ports having the clock option. You typically also want to use it for Async reset input ports.

The differences between a normal input boundary scan cell and one assigned the sample_only option are illustrated in [Table 10-5](#) below. Refer to [Table 10-3](#) for information

on how an inout cell is constructed. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

Table 10-5. Illustration of the “sample_only” Boundary Scan Cell Option

1) This example shows a normal input boundary scan cell. The multiplexer shown on the right intercepts the from_pad connection of the pad cell. The boundary scan cell can drive a value to the core in INTEST mode.

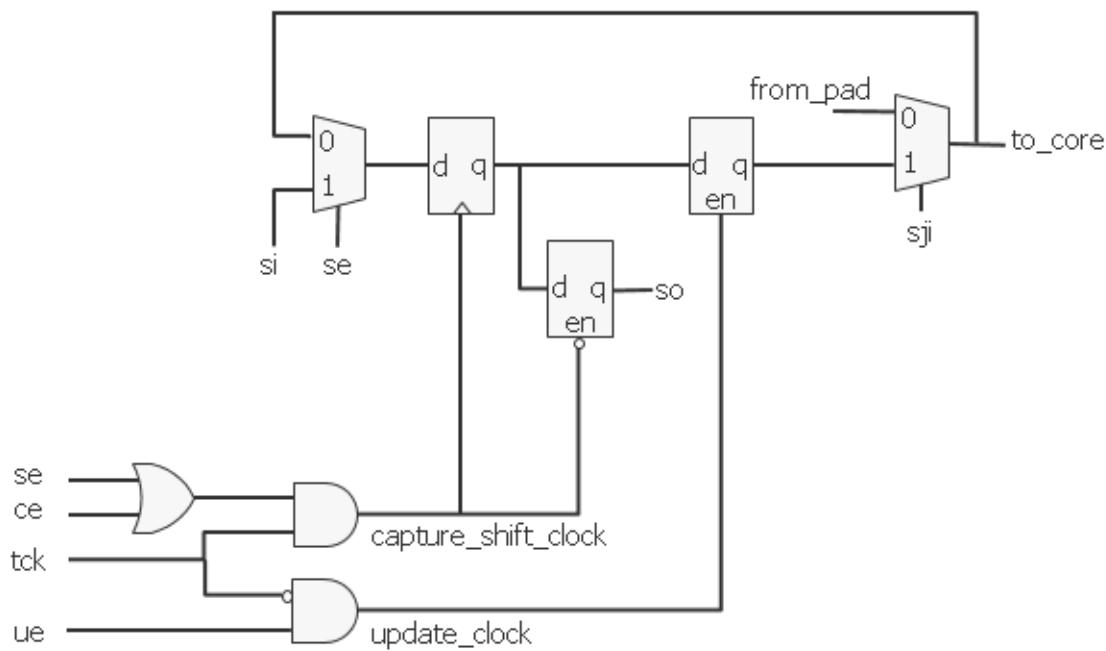
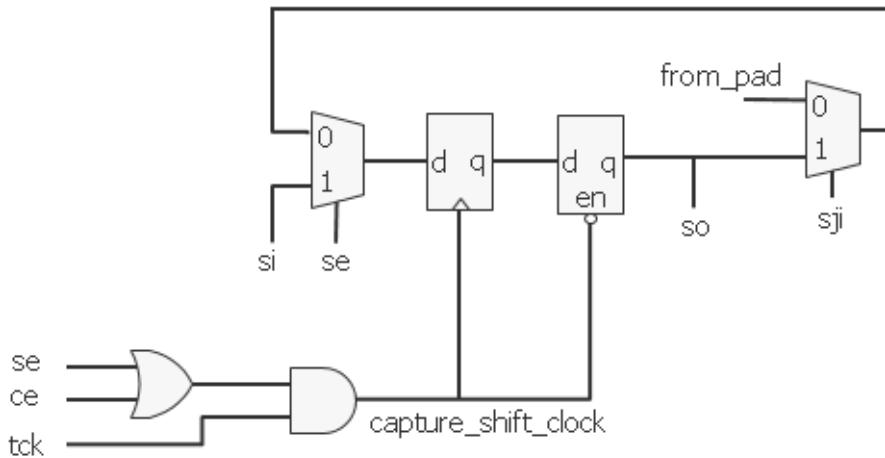


Table 10-5. Illustration of the “sample_only” Boundary Scan Cell Option (cont.)

2) The example below shows a sample_only input boundary scan cell. The multiplexer shown on the right will no longer intercept the from_pad connection of the pad cell, but will simply connect to it, preserving the direct connection between the pad and core. Note that compared to the normal input boundary scan cell, the update latch has been removed and the retiming latch is now connected to the multiplexer on the right.



Examples

The following example uses the `BoundaryScanCellOptions` wrapper to identify VDD and VSS ports in the BSDL file. These port do not exist in the design. The pattern `clk*` is used to match any ports in the design starting with the string “clk” as clock ports. They will not have a JTAG mux inserted along their functional input path, and they will be identified as Clock ports in the BSDL file.

```
DftSpecification(mychip,rtl) {
    BoundaryScan {
        BoundaryScanCellOptions {
            VDD%d[0:32] : power ;
            VSS%d[0:56] : ground ;
            clk* : clock;
        }
    }
}
```

LogicalGroups

Specifies how to group the boundary-scan cells into logical grouping modules.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        LogicalGroups {
            LogicalGroup(group_name) {
                first_port : port_name ;
                parent_instance : parent_instance ;
                leaf_instance_name : leaf_instance_name ;
            }
        }
    }
}
```

Description

A wrapper that specifies how to group the boundary-scan cells into logical grouping modules. You can specify more than one LogicalGroup wrapper only when you have specified the `pin_order_file` property in the [BoundaryScan](#) wrapper.

You use the `first_port` property to identify the first port in the pin order file that belongs to the logical grouping. The association between ports and the LogicalGroup can also be specified in the forth column of the pin order file. If you do not specify any `LogicalGroup(group_name)` wrapper, all boundary scan cells are grouped into a single module, the module is instantiated into the top module, and the leaf instance name is as follows:

`design_name_design_id_tessent_bscan_logical_group_DEF_inst`

as if you had specified an empty `LogicalGroup(DEF)` wrapper.

You use the `parent_instance` property to specify where you want the logical grouping module to be instantiated. By default, it will be instantiated in the top module. You use the `leaf_instance_name` property to specify the leaf instance name used when instantiating the module if you are not satisfied with the default value. The default `leaf_instance_name` used is `design_name_design_id_tessent_bscan_logical_group_group_name_inst`.

The example below shows how to specify the `parent_instance` property in order to insert each logical grouping module in the same instance where the pins or nets to be intercepted by the boundary-scan cells are located.

Arguments

- `group_name`

A string that uniquely identifies each logical group. The string starts with a letter and it is allowed to be followed by any number of letters, numbers, and underscores. The `group_name` is used in constructing the grouping module name as follows:

`design_name_design_id_tessent_bscan_logical_group_group_name`

- `first_port : port_name ;`

A property that specifies a port name that exists in the specified pin_order_file. This property is optional when there is only one LogicalGroup wrapper but it is required when there are more than one. Specifying a port_name for the first_port property is equivalent to specifying the group_name in the forth column of that port in the pin_order_file. See the pin_order_file property description in the BoundaryScan wrapper for a description of that file format and usage.

- `parent_instance : parent_instance ;`

A property that controls where the logical grouping module will be instantiated. By default, it is instantiated in the root module but you can push it down into instances. See the following example to understand the requirement that must be met when choosing the parent_instance of the logical grouping module.

If the specified parent_instance is not unique, it will be uniquified before the logical grouping module is inserted in it. You can do bottom up insertion using the [EmbeddedBoundaryScan](#) wrapper on sub modules to insert segments of boundary scan chains into repeated modules, and then reuse them as building blocks at the next level up if you want to avoid uniquifying the parent_instance. Specify the “[set_design_level sub_block](#)” command and option when inserting boundary scan into those sub modules.

- `leaf_instance_name : leaf_instance_name ;`

A property that specifies a different leaf name when instantiating the logical grouping module into its parent instance. The default value when unspecified is `design_name_design_id_tessent_bscan_logical_group_group_name_inst`.

If you choose to change it, you do not have to worry about picking a unique leaf name in the given parent_instance. The tool will automatically append the `instance_uniquification_suffix` to your specified leaf instance name in case of conflicts. See the [set_insertion_options](#) command description for a complete description of the `-instance_unification_suffix` option.

Examples

The following example illustrates the use of the parent_instance property of the LogicalGroup wrapper to meet the rule that the logical grouping module must be instantiated where all the intercepted nets are visible. [Figure 10-22](#) shows the circuit before the insertion of the boundary-scan cells. You can see that the functional connections to and from the pad cells associated with ports P3 and P4 do not exist outside the B1_I instance. In this example, a single logical grouping module is specified so it can only be instantiated inside B1_I as illustrated in [Figure 10-23](#).

```
BoundaryScan {
    LogicalGroups {
        LogicalGroup(g1) {
            parent_instance : B1_I ;
        }
    }
}
```

The same example is modified to use a pin_order file thus enabling the splitting of the boundary-scan cells into more than one logical grouping module. The pin order file defines the order from P1 to P4. Two logical grouping modules are specified in which the first one, g1, starts with P1 and goes until P2 and the second one, g2, starts with P3 and goes until P4. The g2 group is still restricted to having its parent_instance property as B1_I because this is the only place where all of the connections to be intercepted are visible. The g1 group is, on the other hand, allowed to be instantiated in four locations because all of the connections to be intercepted are visible in those four locations. The parent_instance is shown for the four options and the corresponding locations of the logical grouping modules are shown in [Figure 10-24](#).

```
BoundaryScan {
    pin_order_file      : mychip.pin_order
    LogicalGroups {
        LogicalGroup(g1) {
            first_port      : P1 ;
            parent_instance : B1_I ;           // Option1
            parent_instance : . ;             // Option2
            parent_instance : B2_I ;           // Option3
            parent_instance : B2_I/B3_I ;     // Option4
        }
        LogicalGroup(g2) {
            first_port      : P3 ;
            parent_instance : B1_I ;
        }
    }
}
```

The example also illustrates that you can specify the outputs_per_enable_cell property with a value smaller than the actual fanout count of the functional enable signal. In this case, the functional enable signal fans outs to two output pads. One enable boundary-scan cell is inserted per output port, requiring the tool to split the output enable connectivity to each pad.

[Figure 10-25](#) shows this example with option1 for the parent_instance property. No new ports need to be created. The functional enable signal is observable by two enable bscan cells, and one connection per output pad is made between the logical grouping module and the enable pin of the output pad cells.

[Figure 10-26](#) shows this example with option3 for the parent_instance property. In this case, the connections between the logical grouping module and the output pads require the creation of new pins as illustrated with the orange net crossing the hierarchy. If you are familiar with Tessent Boundary Scan within ETAssemble, the flexibility to precisely control the fanout of the enable signals was not present in that tool. All of those existing limitations are removed within Tessent Shell, and a net driven by RTL can now be intercepted and split for boundary scan usage. In ETAssemble, you needed the functional enable signal to be identified on a pin which often meant you had to insert buffers in the RTL to enable the Boundary Scan to be placed in the same module as the pads.

```
BoundaryScan {
    pin_order_file      : mychip.pin_order
    OutputPerEnableCell : 1 ;
    LogicalGroups {
        LogicalGroup(g1) {
            first_port      : P1 ;
            parent_instance : B1_I ;           // Option1
            parent_instance : B2_I ;           // Option3
        }
        LogicalGroup(g2) {
            first_port      : P3 ;
            parent_instance : B1_I ;
        }
    }
}
```

Figure 10-22. Example Design With Pads

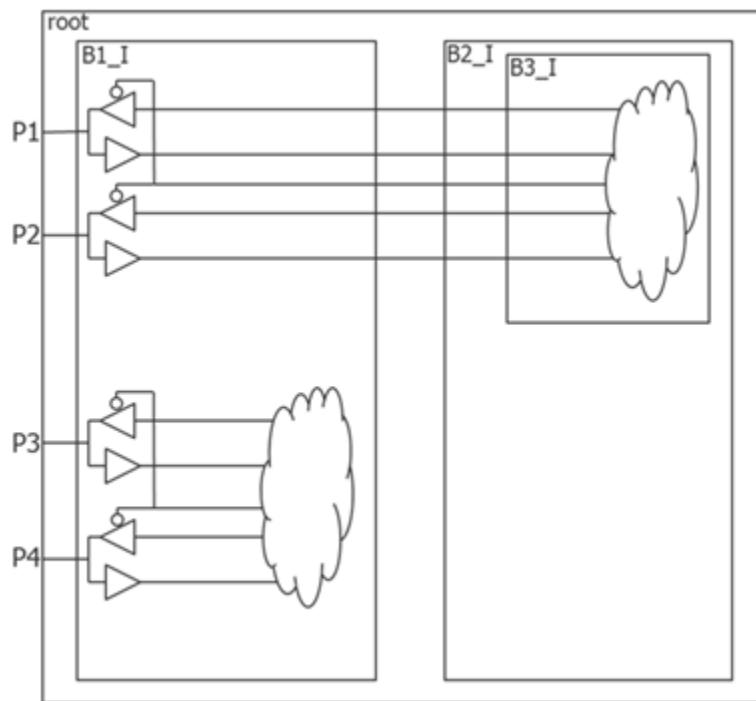


Figure 10-23. Single Logical Group Example

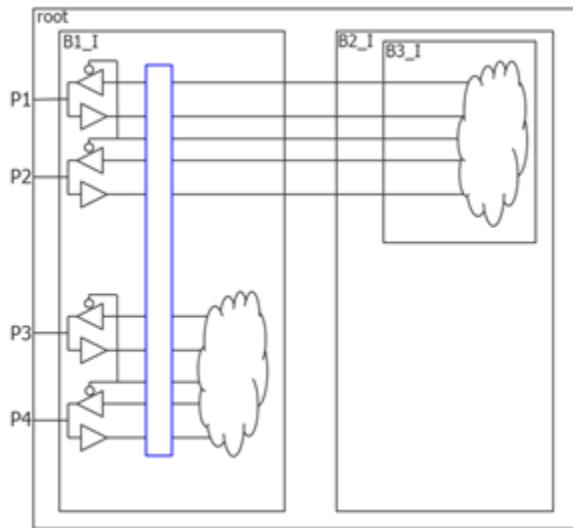


Figure 10-24. Two Logical Groups Example

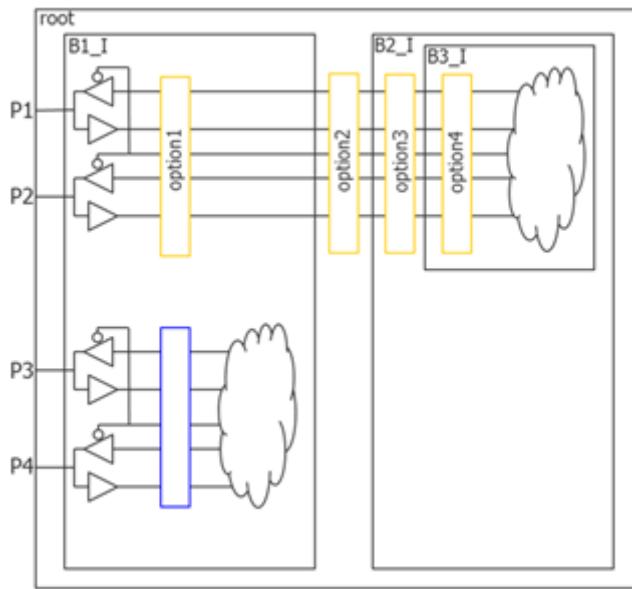


Figure 10-25. Two Logical Groups Example1 with One Output per Enable Cells

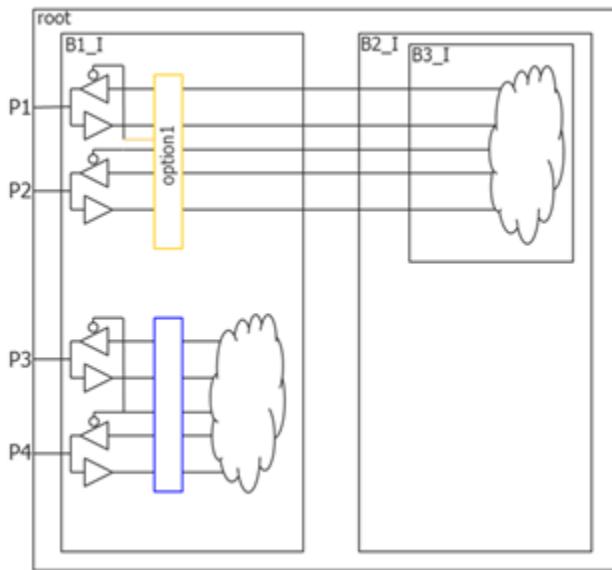
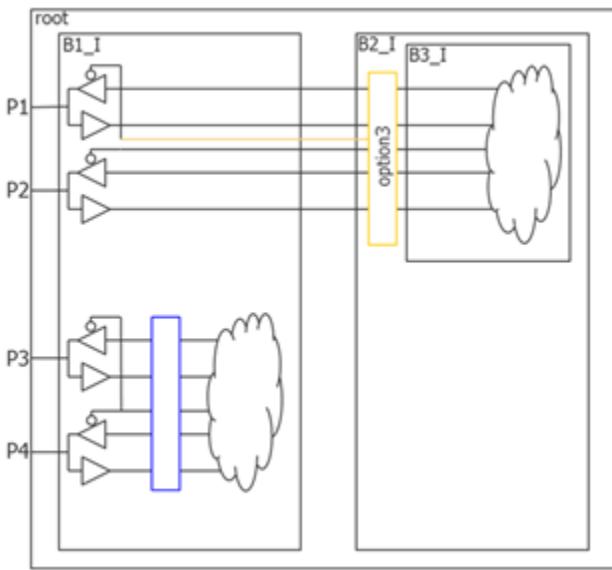


Figure 10-26. Two Logical Groups Example2 with One Output per Enable Cells



ACModeOptions

Specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        ACModeOptions {
            test_receiver_init_clock_value : initialization_event ;
                // Legal: logic_low logic_high
                // falling_edge rising_edge auto
            pulse_min_duration : time | auto ;
            train_minimum_count : count | unused ;
            train_max_duration : time | unlimited ;
            ACGroup(group_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                port_list : port_name, ... ;
            }
        }
    }
    EmbeddedBoundaryScan {
        ACModeOptions {
            test_receiver_init_clock_value : initialization_event ;
                // Legal: logic_low logic_high
                // falling_edge rising_edge auto
            ACGroup(group_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                port_list : port_name, ... ;
            }
        }
    }
}
```

Description

A wrapper that specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain. You use this wrapper to define AC enable cells and associate them with a group of ports. You also use this wrapper to specify the event type that initializes the test receiver when the value is not described in the cell library. Finally, you can use this wrapper to specify three properties that are simply forwarded into the BSDL file.

Arguments

- test_receiver_init_clock_value : initialization_event ;

A property that is useful when the cell library description of the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the new attributes that precisely describe the initialization event: pad_init_posedge_clock_dot6, pad_init_negedge_clock_dot6, pad_init_enable_high_dot6, and pad_init_enable_low_dot6. If the value is “auto” and the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the four new attributes, the tool issues an error; you are required to specify the initialization event when it cannot be inferred from the cell library.

- `pulse_min_duration : time | auto ;`

An optional property that defines the minimum time to use between the rising and a falling edge during the AC tests. This property is forwarded into the BSDL file using the AIO_EXTEST_Pulse_Execution attribute. When unspecified or specified to “auto”, the minimum pulse duration is chosen so as not to be smaller than three times the larger of the HP_time and LP_Time BSDL values of each pad cell.

```
attribute AIO_EXTEST_Pulse_Execution of <current_design>: entity is
    "Wait_Duration <pulse_min_duration_in_seconds>" ;
```

- `train_minimum_count : count | unused ;`

An optional property that defines the minimum number of pulses the train test must have. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. The property defaults to “unused” when unspecified. Specifying a value of one effectively makes the train test equivalent to the pulse test.

```
attribute AIO_EXTEST_Train_Execution of <current_design>: entity is
    "train <train_minimum_count>, maximum_time <train_max_duration>" ;
```

- `train_max_duration : time | unlimited ;`

An optional property that defines a maximum time the train test must take. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. It defaults to “unlimited” when unspecified.

- `group_name`

A string that uniquely identifies the ACGroup. This name is not reflected in the BSDL file but is used to name the leaf instance name of the AC group enable cell in the logical grouping module.

If no AcGroup wrappers are specified, then all AC ports are active during the AC tests. If you specified at least one ACGroup wrapper but you have some AC ports not associated with an ACGroup, they will always be active during the AC tests. In this situation, a warning is generated to let you know that not all AC pins were equipped with an AC enable cell. You can define an extra ACGroup wrapper without a port_list property to indicate that you want all AC ports not associated with another ACGroup wrapper to be associated with that one.

- `insert_before_port : port_name ;`

A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the [BoundaryScan](#) wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

- `insert_after_port : port_name ;`

A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the `pin_order_file` property inside the [BoundaryScan](#) wrapper is specified. The specification of the `insert_before_port` property is mutually exclusive with the specification of the `insert_after_port` property. If neither the `insert_before_port` nor `insert_after_port` properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the `AIO_Pin_Behavior` attribute.

- `port_list : port_name_pattern, ... ;`

A property used to list the port names that belong to the AC group. The `port_name_pattern` can include the asterisk “*” as a wildcard and `%#d[lb:rb]` as a counter. The `#` entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, `ABC%2d[0:2]` will match ABC00, ABC01, and ABC02.

You are allowed to have one `ACGroup` wrapper with an unspecified `port_list`. This `ACGroup` will collect all AC ports that are not already associated with another `ACGroup` wrapper.

Examples

The following example defines an `ACGroup` wrapper with the name “`pci`” to specify that all ports starting with `PCI_` belong to the group. All other AC ports are associated with the “`other`” group. The “`other`” `ACGroup` collects all other AC ports because it does not specify the `port_list` property.

```
DftSpecification(my_chip,rtl) {
    BoundaryScan {
        AcGroups {
            ACGroup(pci) {
                port_list : PCI_* ;
            }
            ACGroup(other) {
            }
        }
    }
}
```

AuxiliaryInputOutputPorts

Specifies two lists of port name patterns to be used as Auxiliary input and/or output ports during some test modes.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        AuxiliaryInputOutputPorts {
            auxiliary_input_ports : port_name_pattern, ... ;
            auxiliary_output_ports : port_name_pattern, ... ;
        }
    }
    EmbeddedBoundaryScan{
        AuxiliaryInputOutputPorts {
            internal_auxiliary_input_ports : port_name_pattern, ... ;
            internal_auxiliary_output_ports : port_name_pattern, ... ;
            external_auxiliary_input_ports : port_name_pattern, ... ;
            external_auxiliary_output_ports : port_name_pattern, ... ;
        }
    }
}
```

Description

A wrapper that specifies two lists of port name patterns to be used as Auxiliary input and/or output ports during some test modes. An inout port is allowed to be specified in both lists at the same time. In the output directions, a 2-to-1 multiplexer (shown in green in [Figure 10-29](#)) is added on the test side of the JTAG multiplexer (shown in red in [Figure 10-29](#)). The auxiliary output multiplexing is added in the boundary-scan cell to avoid cascading two multiplexers along the functional output path. For auxiliary input logic, an AND gate is used to prevent the aux_in_data signal from toggling in functional mode. The aux_in_en signal is also used to force the output pad driver off when adding AuxIn logic on an inout pad.

The aux_out_enable and aux_in_enable signals are kept separate for each port to provide full flexibility as to which output and input ports are enabled to carry input or output auxiliary data in any given test mode. The ports equipped with auxiliary data input and output logic are useful for supplying channel input and output data to EDT controllers. They are also useful to provide the source of the scan_en, test_clock, and edt_update DFT signals. See the description of the [add_dft_signals](#) command for more info. Notice that the aux_out_en and aux_in_en signals have higher priority than the select_jtag_input and select_jtag_output signals. The reason for this is explained below but has the implication that the aux_out_en and aux_in_en sources must reset off and remain off when using the IEEE 1149.1 test mode of the chip. This requirement is met if you use the DFT signals created by the [add_dft_signals](#) to control those pins.

[Figure 10-27](#) shows the typical boundary scan cell inserted on a inout pad. The red multiplexers are those referred to as JTAG multiplexers in the first paragraph. They are controlled by the select_jtag_output to allow the output data and the output enable values to be controlled by the Boundary Scan cells in IEEE 1149.1 test modes.

Figure 10-27. Typical Boundary Scan Cell Inserted for an Inout Pad

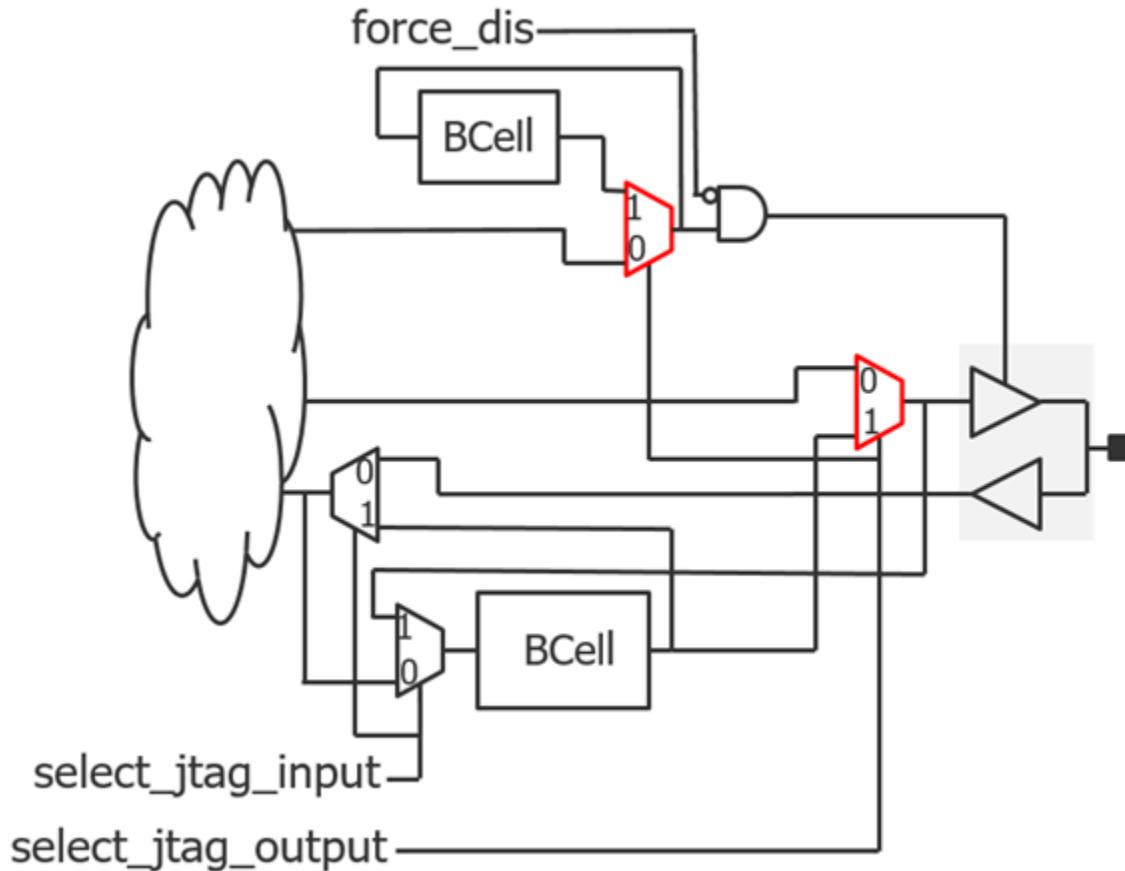


Figure 10-28 shows the boundary scan cell when it is used during ATPG. If you set the `max_segment_length_for_logictest` property found in the Boundary Scan wrapper to an integer or to unlimited, ATPG scan chains are created with the Boundary Scan chain. The blue path shows how the boundary scan cell is used to isolate the core values from the input ports by forcing `select_jtag_input` to 1 which is done by forcing the `int_ltest_en` pin on the boundary scan interface module. For more information about the boundary scan interface module and the logic inserted in it when the boundary scan chain is reused in logic test, see [Figure 10-35](#) on page 3391.

Note

If you are using the DFT signal flow, simply add the `int_ltest_en` DFT signal using the `add_dft_signals` command when you reach the top level and it will be connected to the boundary scan interface module. You can then activate the input isolation using the “`set_static_dft_signal_values int_ltest_en 1`” command and switch when it is time to create your ATPG patterns.

The `select_jtag_output` signal is under ATPG control allowing both the input of the output of the JTAG multiplexers (shown in red) to be tested. If you do not force the `int_ltest_en` pin to 1, then the `select_jtag_input` will also be under ATPG control and the input path can also be tested.

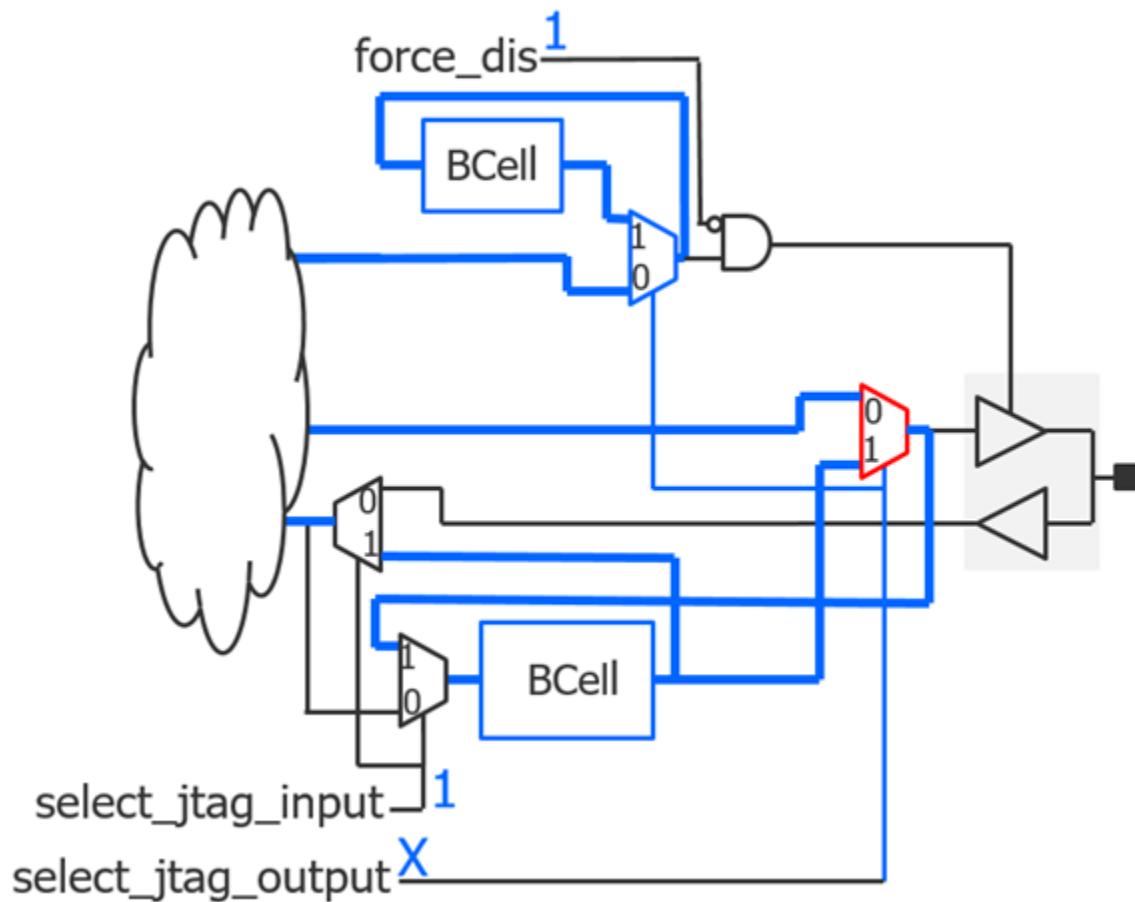
Figure 10-28. Boundary Scan Cell as Observation and Control in Logic Test Modes

Figure 10-29 shows the same boundary scan cell shown in Figure 10-27 but with auxiliary logic (colored green) added for both the input and output directions. Notice how the green multiplexer is not inserted along the functional path but on the test side of the JTAG multiplexer shown in red. The `aux_in_en` and `aux_out_en` signals are mutually exclusive, and the boundary scan cell is in functional mode when they are both low.

Figure 10-29. Adding Auxiliary Input and Output Logic in Bscan Cell

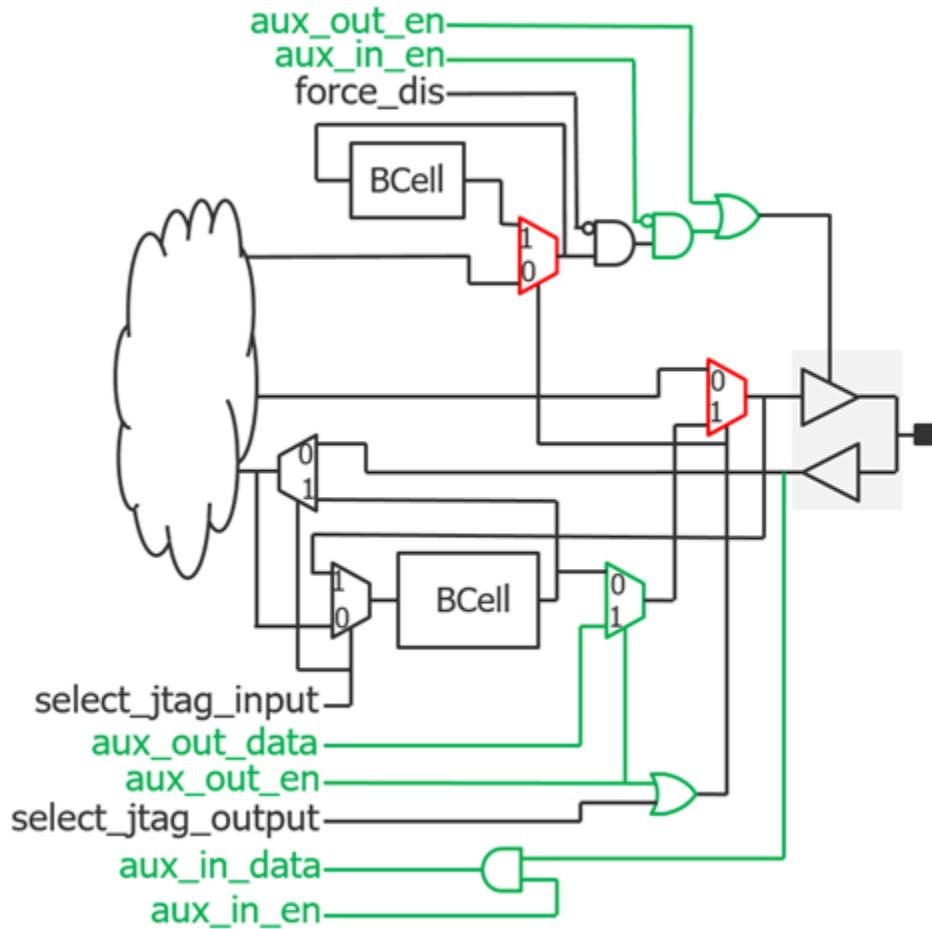


Figure 10-30 shows aux_out_en asserted to 1 to enable the auxiliary output data path. The two multiplexers are switched to select the aux_out_data value, and the enable of the output pad is forced on. When aux_out_en is high, the boundary scan cell is not able to observe the value coming from the core. This is why the [add_dft_modal_connections](#) command gates the aux_out_en signals with “scan_en” such that it is active only in shift mode. In capture mode, the aux_out_data is not needed, and instead the boundary scan cell is used to observe the functional core output. The capture mode is shown in Figure 10-31.

Figure 10-30. Path Used When AuxOut Logic Activated

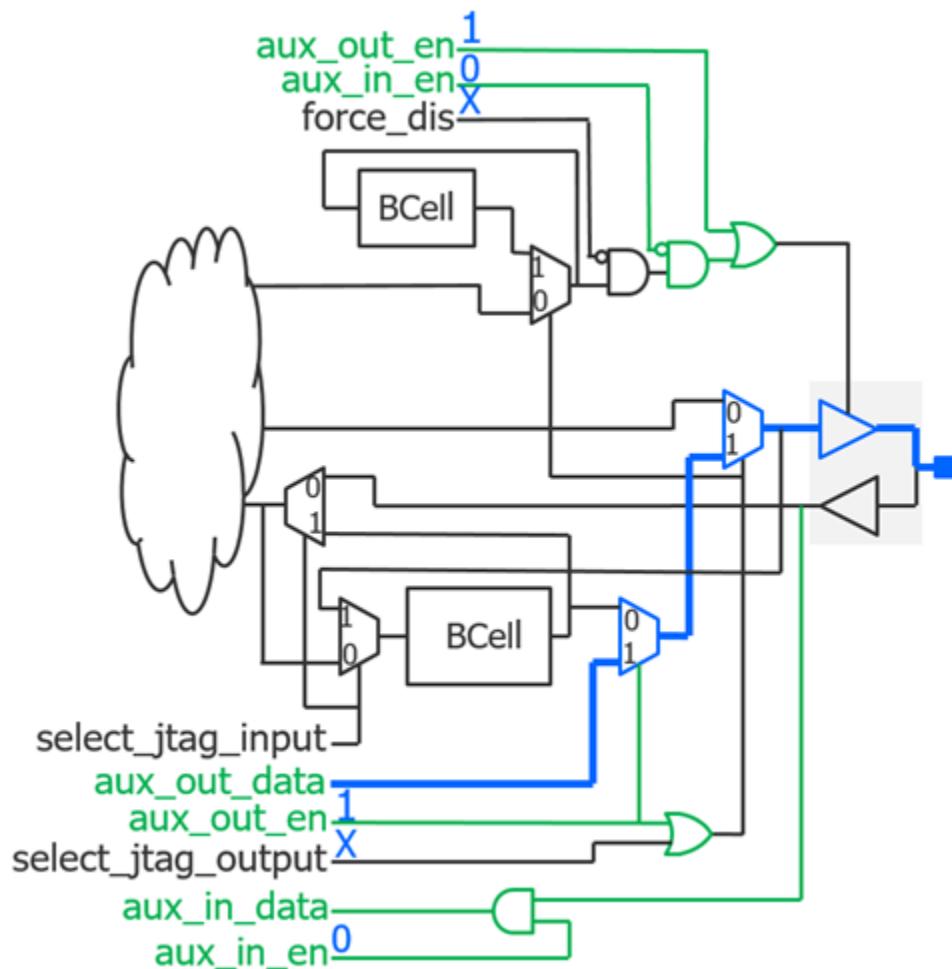


Figure 10-31. Forcing aux_out_en Low in Capture Mode to Observe Core Output

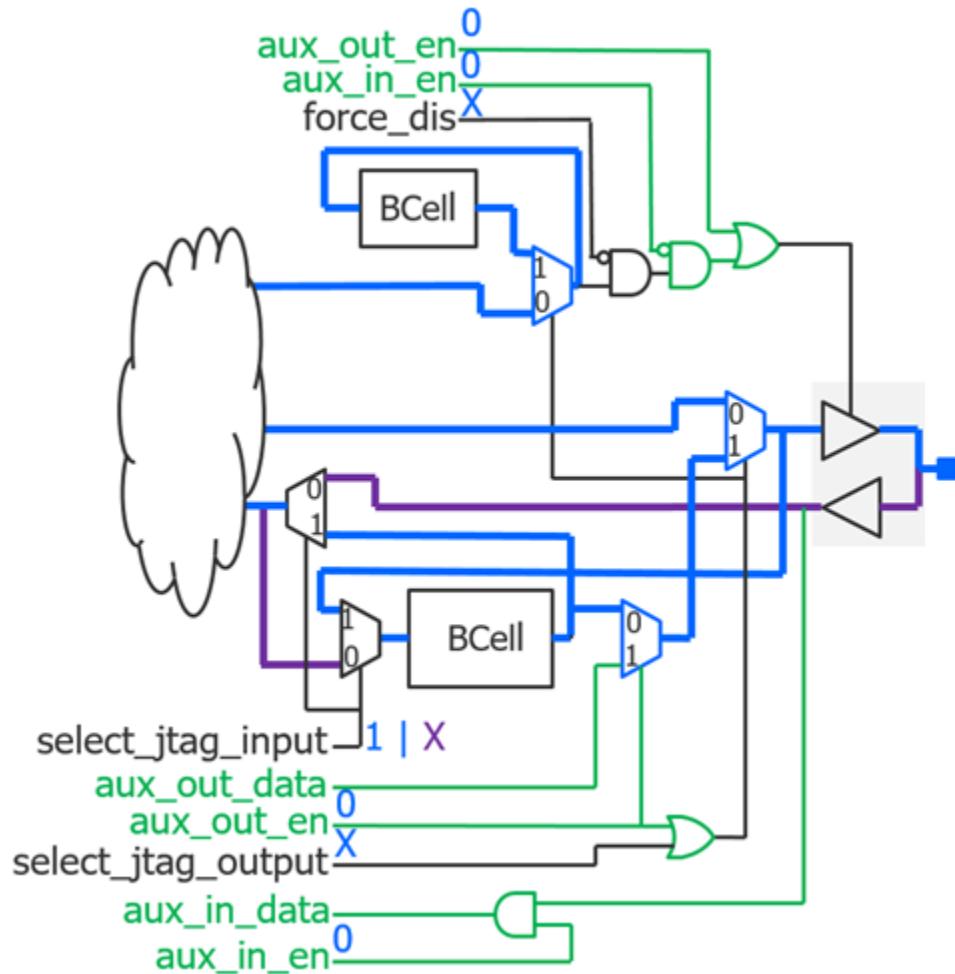


Figure 10-32 shows the IO pad used as an auxiliary input during ATPG. The boundary scan cell is used to control the signal going to the core while the input buffer provides the value to the auxiliary in data. This scheme is useful to allow reusing functional pins for the DFT signals scan_en, test_clock, and edt_update without losing ATPG fault coverage of those inputs.

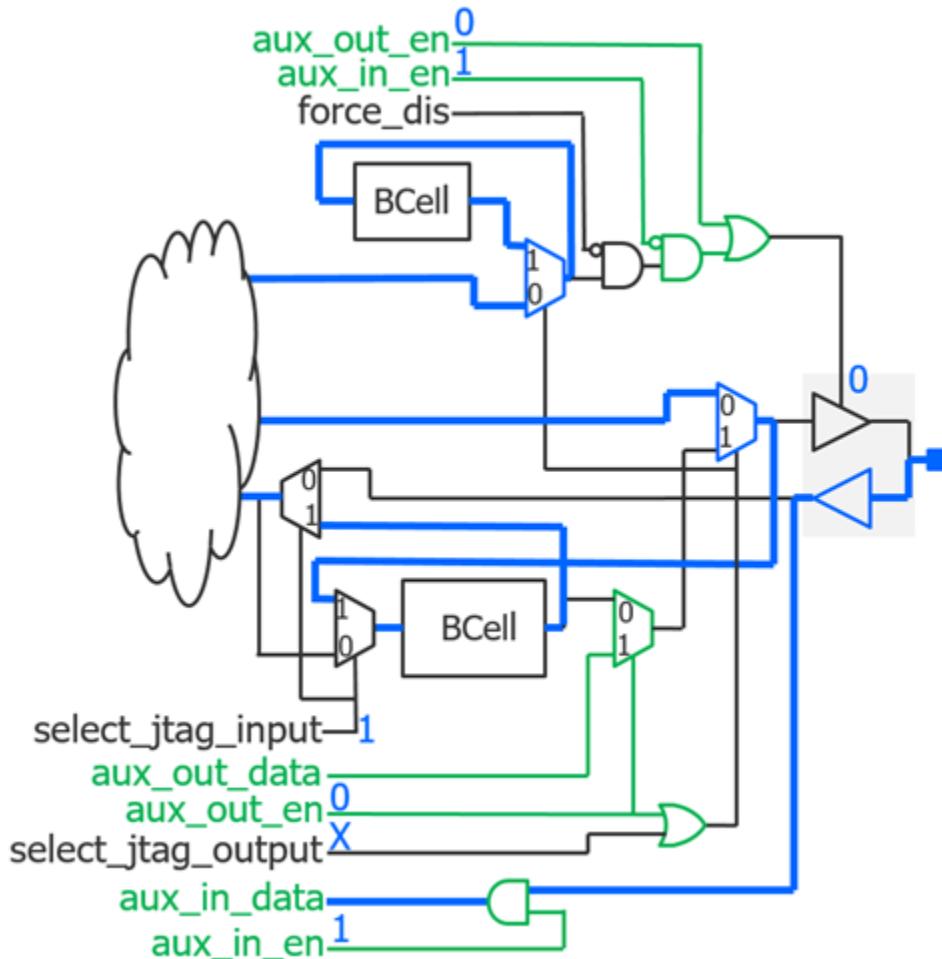
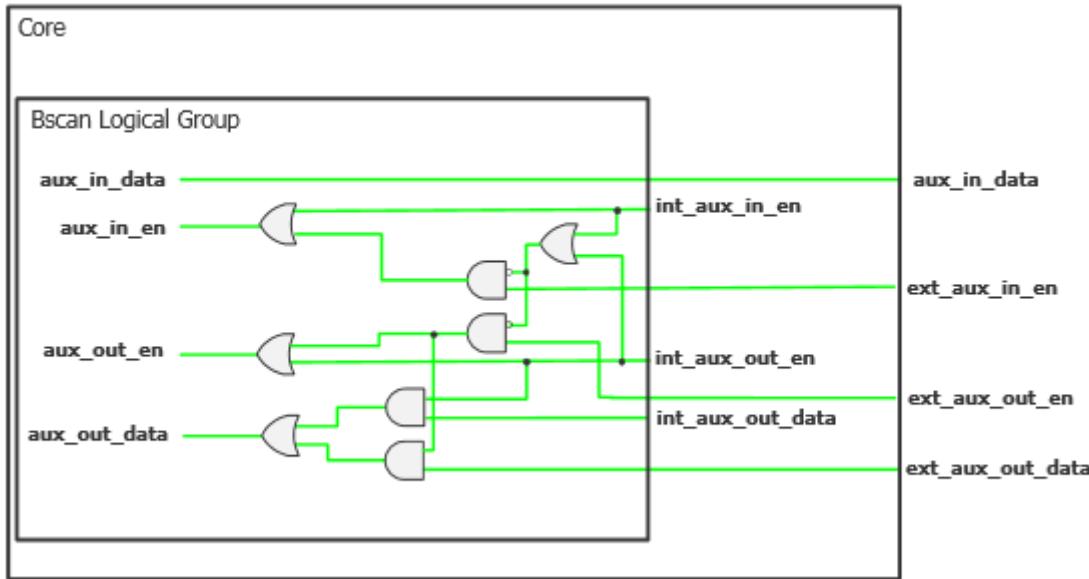
Figure 10-32. Using AuxIn and Controlling to Core Signal in ATPG

Figure 10-33 shows the extra logic that is added when the same port is used as both an internal and an external auxiliary data in the EmbeddedBoundaryScan/AuxiliaryInputOutputPorts wrapper. In the output direction, two data_out input pins are created on the logical grouping module. One of those inputs is used to make auxiliary output connections for elements found within the block and the other input is connected to an input of the block to make auxiliary output connections for elements found outside the block. In the input direction, the same data output port on the logical grouping module is used for providing auxiliary data inputs to elements located inside and outside the block. Both directions have an internal and external enable pin. The internal pins are used by the elements inside the blocks and the external pins are used by the elements outside the blocks. For example, if you call `add_dft_modal_connections` inside the block, it will make connections to the internal pins; and if you call `add_dft_modal_connections` outside the block, it will make connections to the external pins.

Figure 10-33. Extra Logic Added When a Port is Used Both Internally and Externally



Arguments

- auxiliary_input_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns to define a group of input or inout ports as auxiliary input ports. The same inout port can be both an input and an output auxiliary port. The auxiliary inputs are left tied on the logic group module to be used by test logic found inside the chip.

The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional, and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- auxiliary_output_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns that defines a group of output or inout ports as auxiliary output ports. The same inout port can be both an input and an output auxiliary port. The auxiliary output ports are left dangling on the logic group module to be used by test logic found inside the chip.

The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- internal_auxiliary_input_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns to define a group of input or inout ports as internal auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the

auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input and output ports of the block to be used by test circuit outside the block.

The *port_name_pattern* values can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `internal_auxiliary_output_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns that defines a group of output or inout ports as internal auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `external_auxiliary_input_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns to define a group of input or inout ports as external auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary output ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the [Interface](#)/`auxiliary_*` properties.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `external_auxiliary_output_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns that defines a group of output or inout ports as external auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the [Interface](#)/`auxiliary_*` properties.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

Examples

The following example defines the GPIO00 to GPIO31 ports as both auxiliary input and output ports. The first 16 are usable by test logic that is both internal and external to the block. The last 16 are only used by test logic that is external to the block. The full flexibility is preserved such that each test mode will be able to use N of them as auxiliary output ports and the rest as auxiliary inputs where N can be different for each test mode.

```
DftSpecification(my_chip,rtl) {
    EmbeddedBoundaryScan {
        AuxiliaryInputOutputPorts {
            internal_auxiliary_input_ports : GPIO%2d[0:15] ;
            internal_auxiliary_output_ports : GPIO%2d[0:15] ;
            external_auxiliary_input_ports : GPIO%2d[0:31] ;
            external_auxiliary_output_ports : GPIO%2d[0:31] ;
        }
    }
}
```

EnableGroups

Specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        EnableGroups {
            EnableGroup(name) {
                capture_core_signal : on | off ;
                internal_enable_signal : pin_or_net_name ;
                port_list : port_name_pattern, ... ;
                insert_before_port : port_name ;
                insert_after_port : port_name ;
            }
        }
    }
}
```

Description

A wrapper that specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list. The properties `insert_before_port` and `insert_after_port` are used to specify the location of the enable cells associated with the group. The property `capture_core_signal` is used to prevent the cell from capturing the core signal during INTEST or during logictest when the boundary scan is used as isolation.

Arguments

- `capture_core_signal : on | off ;`

A property that specifies if the enable boundary-scan cells associated with the group are allowed to capture the core signal during INTEST or during the capture pulse when you reuse it for logictest isolation. See the `max_segment_length_for_logictest` property in the [BoundaryScan](#) wrapper for information about the logictest usage. When specified to off, the enable boundary-scan cell captures itself instead of capturing the core signal. This may be useful if you are using LogicBIST and the functional enable is X during the logictest mode.

- `internal_enable_signal : pin_or_net_name ;`

A property that references an internal signal inside the chip that directly controls the enable pin of a group of output pad cells. Specifying this enable signal is equivalent to listing all the ports controlled by that enable signal in the `port_list` property. The specification of the `internal_enable_pin` and `port_list` properties are mutually exclusive. If the number of output pad cells controlled by the enable signal is larger than the specified `outputs_per_enable_cell` property, the fanout of the enable cell will be split into smaller groups to meet the specified `outputs_per_enable_cell` value. One enable boundary-scan cell will be added per group.

- `port_list : port_name_pattern, ... ;`

A property that associates a group of output or inout ports with a given enable group. The `port_name_pattern` values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a

counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells specified in the list is larger than the specified outputs_per_enable_cell value, the list will be split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell will be added per group.

You do not need to specify an enable group for all output and inout ports. For the tri-state inout and output ports not explicitly associated to an enable group, one or more additional enable groups will be inferred. The inferring is done to match the common functional enable signal sourcing a group of pads. A given group can further be split to satisfy the specified outputs_per_enable_cell value described in the following sections:

- BoundaryScan — [outputs_per_enable_cell](#) property
- EmbeddedBoundaryScan — [outputs_per_enable_cell](#) property
- insert_before_port : *port_name* ;
A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.
- insert_after_port : *port_name* ;
A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

Examples

The following example defines three enable groups. The first one is associated with the ports in the fanout of the functional enable signal “core_i/en”. The second one is associated with all ports matching the pattern “OUT*”. The third group is associated with all other remaining ports as it does not specify either the internal_enable_signal nor the port_list property.

```
DftSpecification(my_chip,rtl) {
    BoundaryScan {
        EnableGroups {
            EnableGroup(g1) {
                internal_enable_signal : core1/en ;
            }
            EnableGroup(g2) {
                port_list : OUT* ;
            }
            EnableGroup(g3) {
            }
        }
    }
}
```

InternalBScanCells

Specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBScanCells {
            InternalBScanCell(name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                safe_value : 0 | 1 | X ;
                type : control | observation | both ;
                connection : pin_or_net_name ;
                multiplexing : on | off | auto ;
            }
        }
    }
}
```

Description

A wrapper that specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Arguments

- *name*

A string that uniquely identifies the internal boundary-scan cell in the BSDL file. The string starts with a letter and may be followed by any numbers of letters, numbers, or underscores. The string must be unique across all InternalBScanCell wrappers. The name of internal boundary-scan cells are mapped to the cell position inside the boundary scan chain using an attribute called TESSENT_INTERNAL_CELL_LABELS as shown in this example. The two InternalBScanCell wrappers have the names “my_name1” and “my_name2”.

```
-- BSDL local extensions
attribute TESSENT_INTERNAL_CELL_LABELS: BSDL_EXTENSION;
attribute TESSENT_INTERNAL_CELL_LABELS of TOP: entity is
    -- Label Cell
    "((my_name1 , 36 ) , " &
    " (my_name2 , 30 ))";
```

- *insert_before_port : port_name ;*

A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the *insert_before_port* property is mutually exclusive with the specification of the *insert_after_port* property. If neither the *insert_before_port* nor *insert_after_port* properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near *scan_out* in the order they were specified. The location

of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- `insert_after_port : port_name ;`

A property that specifies the position of the internal boundary-scan cell. This property can only be used when the `pin_order_file` property inside the [BoundaryScan](#) wrapper is specified. The specification of the `insert_after_port` property is mutually exclusive with the specification of the `insert_before_port` property. If neither the property `insert_before_port` nor `insert_after_port` properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near `scan_out` in the order they were specified. The location of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- `safe_value : 0 | 1 | X ;`

A property that specifies a safe value for the internal boundary-scan cell. This value is reflected into the BSDL file. If a non-X safe value is specified, the [BoundaryScan](#) signoff and manufacturing pattern set will never load a different value into that register during any of its [RunTest](#) specifications.

- `type : control | observation | both ;`

A property that specifies if the internal boundary-scan cell is to be used to control and/or observe an internal node. [Table 10-6](#) shows the hardware inserted into the design based on the value of the type and multiplexing property.

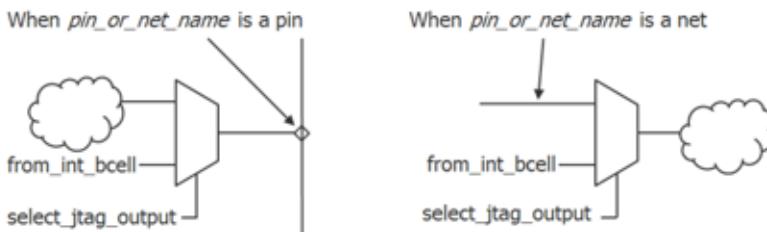
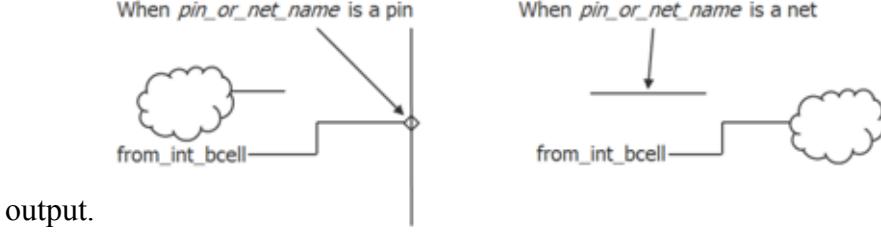
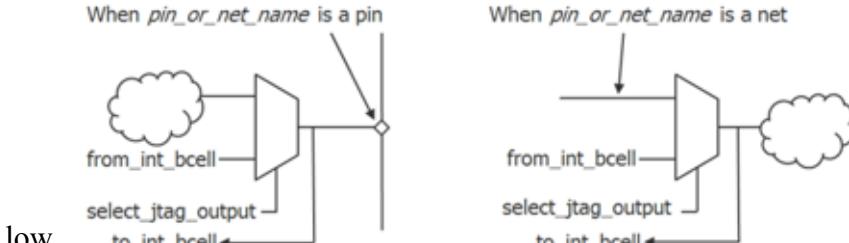
- `connection : pin_or_net_name ;`

A property that specifies the name of a pin or a net to control and/or observe. The `parent_instance` of `pin_or_net_name` will be auto-uniquified if not unique.

- `multiplexing : on | off | auto ;`

A property that is only relevant when the property `type` is set to “control” or “both”. A multiplexer is inserted as shown in [Table 10-6](#). When the value is “auto”, the multiplexer is inserted only when the `has_functional_source` attribute of the specified `pin_or_net_name` is set to “true”. See [Pin](#) and [Net](#) for more information about this attribute.

Table 10-6. Connections to and from Internal Boundary Scan Cells

type	description
control	<p>If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “true”, the driver is intercepted with a multiplexer located in the <i>parent_instance</i> of the <i>pin_or_net_name</i>. A net is intercepted at its source while a pin is intercepted at its destination. The internal boundary-scan cell captures its own output.</p>  <p>If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.</p> 
observation	<p>The specified <i>pin_or_net_name</i> is connected to the input of the internal boundary-scan cell.</p>
both	<p>If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “true”, the driver is intercepted with a multiplexer located in the <i>parent_instance</i> of the <i>pin_or_net_name</i>. A net is intercepted at its source while a pin is intercepted at its destination. The observation net is taken after the multiplexer. The functional source of the <i>pin_or_net_name</i> is observable when <i>select_jtag_output</i> is low.</p>  <p>If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.</p>

Examples

The following example creates an internal boundary-scan cell that is used to control the u1/pll_reset signal using the boundary scan chain. The specified safe value of 0 ensures this cell will never be loaded with a 1 during any of the [RunTest](#) in the [PatternsSpecification](#) or with any board test patterns created from the BSDL file.

```
DftSpecification(module_name,id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBScanCells {
            InternalBScanCell(mycell) {
                insert_before_port : PortA ;
                safe_value         : 0 ;
                type              : control ;
                connection        : u1/pll_reset ;
                multiplexing      : auto ;
            }
        }
    }
}
```

InternalBscanSegments

Specifies the position of an existing internal boundary scan segment in the boundary scan chain.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBscanSegments {
            InternalBScanSegment(instance_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                logical_group_name : logical_group_name ;
            }
        }
    }
}
```

Description

A wrapper that is used to specify the position of an existing internal boundary scan segment in the boundary scan chain. When the BScanSegment contains cells associated with a port of the design, the position of the BScanSegment and its *logical_group_name* is taken from the position and the logical group of the first port associated with the BScanSegment in the *pin_order_file*. In the case of pure internal boundary scan segments, their position and logical grouping name cannot be extracted from associated ports because they are not associated with ports. The logical group name is not used to create logical grouping modules for existing boundary scan segments because they are already grouped into modules. Instead, they are used to refer to the segments that use the *bypass_logical_group* inside the [BondingConfigurations](#) wrapper.

This wrapper is automatically created when the [create_dft_specification](#) command is invoked if a BScanSegment description is found for a module. You simply need to delete this wrapper prior to invoking [process_dft_specification](#) if you do not want it to be part of the boundary scan chain. Refer to the “[set_design_sources -format bscan_lib](#)” command description for information about automatically finding the BScanSegment files without having to explicitly read them in using the [read_core_descriptions](#) command. You specify the module matching options using the [set_module_matching_options](#) command.

Arguments

- *insert_before_port* : *port_name* ;

A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the *insert_before_port* is mutually exclusive with the specification of the *insert_after_port* property. When neither the *insert_before_port* nor *insert_after_port* properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the *pin_order_file* property is used inside the [BoundaryScan](#) wrapper.

- `insert_after_port : port_name ;`

A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the `insert_before_port` is mutually exclusive with the specification of the `insert_after_port` property. When neither the `insert_before_port` nor `insert_after_port` properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the `pin_order_file` property is used inside the `BoundaryScan` wrapper.

- `logical_group_name : logical_group_name ;`

A property used to associate a `logical_group_name` to the existing internal boundary scan segment. This name is not used to create a logical grouping module as the existing boundary scan segment is already grouped into a module. It is used instead to refer to the segment using the `bypass_logical_group` property inside the [BondingConfigurations](#) wrapper.

Examples

The following example defines a design instance as an internal boundary scan segment. It is given a logical group name so that it can be bypassed in some bounding options. A `BscanSegment` definition must exist for the module associated with instance u1/u2.

```
DftSpecification(module_name,id) {
    BoundaryScan {
        InternalBScanSegments {
            InternalBScanSegment(u1/u2) {
                logical_group_name      : seg1 ;
            }
        }
    }
}
```

BoundaryScan

Specifies the boundary scan chain to build and optionally insert into the design.

Usage

```
DftSpecification(module_name,id) {
    BoundaryScan {
        ijtag_host_node           : id ;
        pin_order_file            : file_name ;
        interface_parent_instance : parent_instance ;
        outputs_per_enable_cell   : int ;          // default: 16
        max_segment_length_for_logictest : int | unlimited | off ;
        tristate_enable_non_contacted_test_support : on | off ;
        tck_period                 : period ;     // default 100ns
        ImplementationOptions {
        }
        BoundaryScanCellOptions {
        }
        LogicalGroups {
        }
        ACModeOptions {
        }
        AuxiliaryInputOutputPorts {
        }
        EnableGroups {
        }
        InternalBScanCells {
        }
        InternalBScanSegments {
        }
        BondingConfigurations {
        }
        UserInstructions {
        }
    }
}
```

Description

A wrapper that specifies the boundary scan chain to build and optionally insert into the design.

This wrapper can only be specified once in a given DftSpecification wrapper. As illustrated in the syntax above, the BoundaryScan wrapper uses the `ijtag_host_node` property to specify the host scan interface that drives the boundary scan register; this can be a TAP controller specified in the [IjtagNetwork](#) wrapper of the same DftSpecification, or a tap instance already existing in the design. The TAP controller may have been inserted using a prior execution of the [process_dft_specification](#) command, or it may be a third-party TAP controller manually inserted into your design. If you want to use a third-party TAP controller, you must have a valid ICL description for it and it must have the port functions described in section “[Requirements on a TAP to be usable for BoundaryScan](#)” on page 3385.

If you have existing boundary scan segments that are described with BoundaryScan wrappers (see “[Tessent Core Description](#)” on page 3755), the tool automatically recognizes the existing

boundary scan segments in the design and includes them in the boundary scan chain. The legacy *.lvbscan* file format is also natively supported and translated into the current BoundaryScan wrapper syntax when it is loaded into Tesson shell. You can refer to the [set_module_matching_options](#) and “[set_design_sources](#)-format bscan” commands to learn how to have them automatically searched and loaded. You can also use the [read_core_descriptions](#) command to load them explicitly.

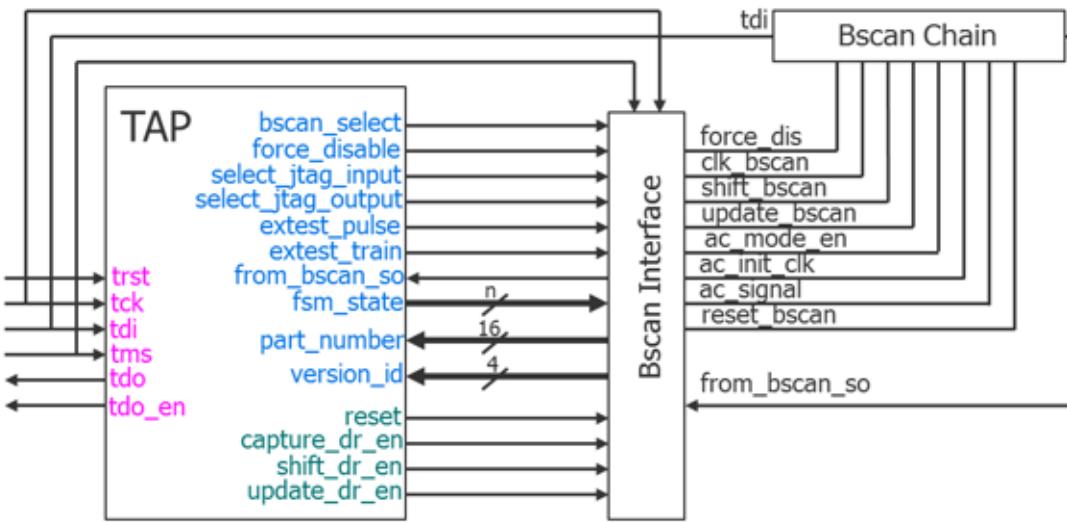
Note

 If you want to insert BoundaryScan and the TAP controller, and connect the TAP to internal pins, follow the instructions in [Example 2](#) of the [IjtagNetwork](#) wrapper description.

Requirements on a TAP to be usable for BoundaryScan

You can use any TAP controller as the access point to the boundary scan as long as it is described in ICL and contains a minimum set of port functions. As with any ICL module that is handed off, the port names used in the ICL file must exactly match the port names of the corresponding Verilog or VHDL module. [Figure 10-34](#) illustrates the connections made to and from a TAP controller to implement boundary scan. The blue ports are ports that are dedicated to Boundary scan. Most of the ports are required. You only need the `extest_pulse` and `extest_train` ports if your chip contains pads that require the IEEE 1149.6 ac modes. You only need the `fsm_state` port if your chip contains pads that require IEEE 1149.6 ac modes or if the chip has `tristate_enable_non_contacted_test_support` testing enabled. You only need the `part_number_code` and `version_code` ports if you want to use the `part_number_code` and `version_code` properties of the [BondingConfigurations](#) wrapper.

Figure 10-34. Needed Resources From Third-Party TAP Controller



The remainder of this section lists the ports that must exist on the third-party TAP controller in order for it to be usable by Tesson BoundaryScan, the conditions that make them required, and

the required ICL descriptions in the ICL files. You can use the following dofile to create a TAP controller and look at its ICL as an example:

```
set_context dft -rtl
read_config_data -from_string {
    DftSpecification(ex,rtl) {
        IjtagNetwork {
            HostScanInterface(tap) {
                Tap(main) {
                    HostBscan {
                    }
                    HostIjtag(1) {
                    }
                    DeviceIDRegister {
                    }
                }
            }
        }
    }
}
process_dft_specification -no_insertion
```

The ICL of the TAP must include the following three attributes: tessent_instruction_reg, tessent_bypass_reg, and tessent_device_id_reg. The value of the attribute is the name of the ScanRegister in ICL that is associated with the instruction, bypass, and dev_id registers. The tessent_device_id_reg attribute is optional and only needed if your TAP supports the device id feature. The ScanRegister referenced by the tessent_instruction_reg attribute must have a RefEnum pointing to an Enum table that defines the listed Opcodes. The IDCODE opcode is optional and only needed if the TAP has a device id register. The EXTEST_PULSE and EXTEST_TRAIN opcodes are only needed if IEEE 1149.6 is used. The tessent_use_in_dft_specification attribute is used to tell the [create_dft_specification](#) command to ignore this module and not infer TDRs to control and observes the unconnected DataIn and DataOut ports of your TAP.

```

Attribute tessent_instruction_reg = "instruction" ;
Attribute tessent_bypass_reg = "bypass" ;
Attribute tessent_device_id_reg = "devid" ;
Attribute tessent_use_in_dft_specification = "false";
ScanRegister instruction[4:0] {
    CaptureSource 5'bXXX01 ;
    ResetValue     5'b00000 ;
    ScanInSource   tdi ;
    RefEnum        IROpcodes ;
}
RefEnum instruction_opcodes {
    BYPASS          = 5'bx0000 ;
    CLAMP           = 5'bx0001 ;
    EXTEST          = 5'bx0010 ;
    EXTEST_PULSE    = 5'bx0011 ;
    EXTEST_TRAIN    = 5'bx0100 ;
    IDCODE          = 5'bx0101 ;
    INTEST          = 5'bx0110 ;
    SAMPLE          = 5'bx0111 ;
    PRELOAD         = 5'bx0111 ;
    HIGHZ           = 5'bx1000 ;
}

```

If your TAP has the part_number_code and/or version_code ports, they must be part of the CaptureSource of the ScanRegister that is referenced by the tessent_device_id_reg attribute as shown here. In this example, man_id is a parameter that defines the 11-bit manufacturing id constant.

```

DataInPort version_code[3:0] {
    Attribute connection_rule_option = "allowed_no_source";
}
DataInPort part_number_code[15:0] {
    Attribute connection_rule_option = "allowed_no_source";
}
ScanRegister devid[31:0] {
    ScanInSource tdi ;
    CaptureSource version_code[3:0], part_number_code[15:0], $man_id, 1'b1 ;
}

```

The BoundaryScan host port must be described in a ScanInterface wrapper as shown here. The actual port name in your TAP may differ from these names, but the name of the ScanInterface must be host_bscan if you want it to be automatically recognized by the [create_dft_specification](#) command without having to use the -existing_bscan_host_scan_in option.

```

ScanInterface host_bscan {
    Port from_bscan_so ;
    Port bscan_select ;
    Port capture_dr_en ;
    Port shift_dr_en ;
    Port update_dr_en ;
}

```

The following list presents the port functions that may exist on the third-party TAP, describes when they are needed, and shows how to describe them in the ICL file. The examples illustrate

the use of the connection_rule_option attribute to indicate to the ICL extraction process that it must allow the bscan host port to *not* be connected to a boundary scan chain. The ICL description of the boundary scan interface module must be connected to a TAP; ICL extraction will report errors if the boundary scan interface module is present but incorrectly connected to a TAP. However, having a TAP controller with no boundary scan chain connected to it is allowed.

- `bscan_en`

This function is required. It is a signal that is high during the EXTEST, SAMPLE, PRELOAD, INTEST, HIGHZ, EXTEST_TRAIN, and EXTEST_PULSE opcodes. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
ToSelectPort bscan_en  {
    Source bscan_en_int;
    Attribute connection_rule_option = "allowed_no_destination";
}
LogicSignal bscan_en_int {
    IR == EXTEST      ||
    IR == SAMPLE     ||
    IR == PRELOAD    ||
    IR == INTEST     ||
    IR == HIGHZ       ||
    IR == EXTEST_TRAIN ||
    IR == EXTEST_PULSE;
}
```

- `force_disable`

This function is required. It is a signal that is active during the HIGHZ opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
DataOutPort force_disable  {
    Source force_disable_int;
    Attribute connection_rule_option = "allowed_no_destination";
    Attribute tessent_bscan_function = "force_disable";
}
```

The value of the attribute tessent_bscan_function is “force_disable_inv” if your port is active low.

- `select_jtag_input`

This function is required. It is a signal that is active during the INTEST opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
    DataOutPort select_jtag_input {
        Source select_jtag_input_int;
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tesson_bscan_function = "select_jtag_input";
    }
```

The attribute tesson_bscan_function is “select_jtag_input_inv” if your port is active low.

- `select_jtag_output`

This function is required. It is a signal that is active during the EXTEST, HIGHZ, CLAMP, EXTEST_TRAIN and the EXTEST_PULSE opcodes. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
    DataOutPort      select_jtag_input {
        Source select_jtag_input_int;
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tesson_bscan_function = "select_jtag_output";
    }
```

The attribute tesson_bscan_function is “select_jtag_output_inv” if your port is active low.

- `extest_pulse`

This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 ac tests. It is a signal that is active during the EXTEST_PULSE opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
    DataOutPort      extest_pulse {
        Source extest_pulse_int;
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tesson_bscan_function = "extest_pulse";
    }
```

The attribute tesson_bscan_function is “extest_pulse_inv” if your port is active low.

- `extest_train`

This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 tests. It is a signal that is active during the EXTEST_TRAIN opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It is must be described in the ICL file as shown here.

```
    DataOutPort      extest_train {
        Source extest_pulse_int;
        Attribute connection_rule_option = "allowed_no_destination";
        Attribute tesson_bscan_function = "extest_train";
    }
```

The attribute tesson_bscan_function is “extest_train_inv” if your port is active low.

- fsm_state

This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 tests or if [tristate_enable_non_contacted_test_support](#):on is specified. It is a bus that represents the TAP fsm state encoding. The fsm_state signals must change on the rising of TCK, and they must be described in the ICL file as shown here. The name of the Enum can be any arbitrary name as long as it includes the symbols for all 16 fsm state values.

```
    DataOutPort      fsm_state[3:0] {
        Attribute connection_rule_option = "allowed_no_destination" ;
        Attribute function_modifier = "tap_fsm_state" ;
        RefEnum      state_encoding ;
    }
    Enum state_encoding {
        test_logic_reset      = 4'b0000 ;
        run_test_idle         = 4'b0001 ;
        select_dr             = 4'b0010 ;
        capture_dr            = 4'b0011 ;
        shift_dr              = 4'b0100 ;
        exit1_dr              = 4'b0101 ;
        pause_dr               = 4'b0110 ;
        exit2_dr              = 4'b0111 ;
        update_dr              = 4'b1000 ;
        select_ir              = 4'b1001 ;
        capture_ir             = 4'b1010 ;
        shift_ir               = 4'b1011 ;
        exit1_ir              = 4'b1100 ;
        pause_ir               = 4'b1101 ;
        exit2_ir              = 4'b1110 ;
        update_ir              = 4'b1111 ;
    }
```

If you want to insert BoundaryScan and the TAP controller, and connect the TAP to internal pins, follow the instructions in “[Example 2](#)” on page 3227 in the [IjtagNetwork](#) wrapper description.

Boundary Scan Interface Module

As shown in [Figure 10-34](#), the boundary scan interface module resides between the TAP controller and the boundary scan chain. The normal logic included in the interface module is shown in black in [Figure 10-35](#) and includes the clock gaters to create the bscan_update_clock and bscan_shift_capture_clock used by some boundary scan cells. See the [ImplementationOptions](#) wrapper’s description section for a description of boundary scan cells using gated clocks. If you are using IEEE 1149.6 boundary scan cells, the interface will have more logic that is used to create the IEEE 1149.6 signals but this logic is not shown in [Figure 10-35](#). The blue logic shown in [Figure 10-35](#) is the logic added in the interface module when the boundary scan chain is re-used during the logic test modes. This feature is requested by setting the “[max_segment_length_for_logictest](#)” property described below to an integer or to “unlimited”. The inserted multiplexer segments the boundary scan chains into one or more logic test scan chains as shown in [Figure 10-36](#). The name of the scan_in and scan_out pins created by this feature are documented in the [Instrument Dictionary](#) described below. The blue module

shown in bold contains a scan chains of 6 flip-flops between `ltest_si` and `ltest_so` used to provide ATPG control to the 6 output ports of the module. This allow achieving high ATPG fault coverage of the boundary scan logic. The same module also include a mini-OCC as shown in [Figure 10-13](#) on page 3269 of the [Sib](#) wrapper description section. Notice how the “`int_ltest_en`” input is ORed with the ATPG controlled “`select_jtag_input`” signal before being sent to the boundary scan cells. This enables using the boundary scan cells to isolate the input of the chip from the core input values in order to support low count pin test for multi-site testing. See [Figure 10-31](#) on page 3370 and [Figure 10-32](#) on page 3371 in the [AuxiliaryInputOutputPorts](#) section for more explanation of this feature. You use the “`set_static_dft_signal_values int_ltest_en 1`” to set `int_ltest_en` in your `test_setup`.

Figure 10-35. Schematic of the Boundary Scan Interface Module

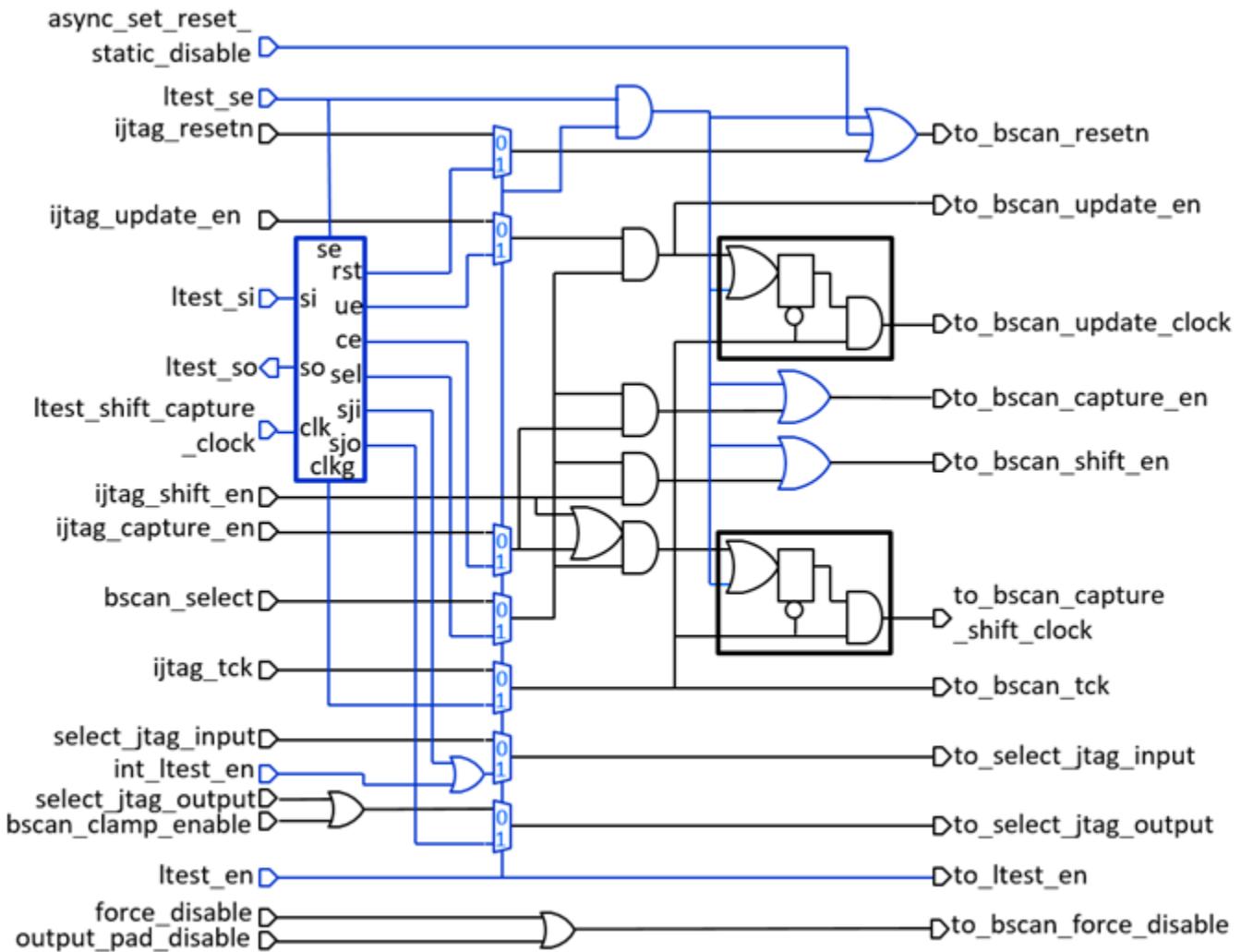
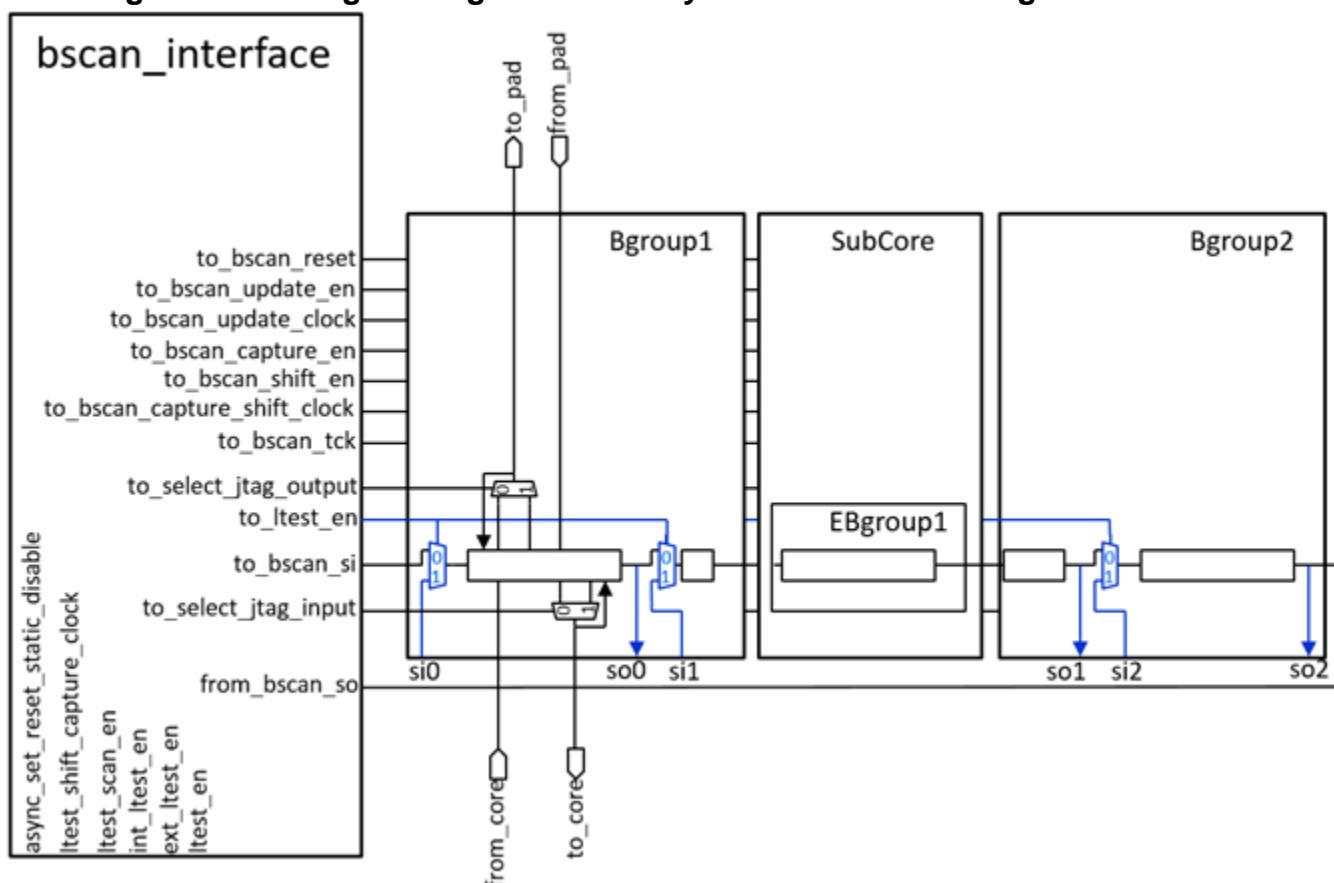


Figure 10-36. Segmenting the Boundary Scan Chains into Logic Test Chains



Instrument Dictionary

When you have run the `process_dft_spec` commands and the `BoundaryScan` wrapper or the `EmbeddedBoundaryScan` wrapper was present in the `DftSpecification`, an instrument dictionary is created to describe what was done on each ports. You can access this dictionary using this command:

```
format_dictionary [get_instrument_dictionary mentor::jtag_bscan::DftSpecification]
```

The format of the dictionary is the following:

```
ports {
    <port_name> {
        bscan_cell_presence added | existing | none
        bcell_pad_connection_functions {
            from_pad          pin_name
            to_core           pin_name
            from_core         pin_name
            to_core           pin_name
            init_data_dot6   pin_name
            init_data_inv_dot6 pin_name
            test_data_dot6   pin_name
            test_data_inv_dot6 pin_name
            from_sji_mux     pin_name
            to_sji_mux       pin_name
            from_sjo_mux     pin_name
            to_sjo_mux       pin_name
        }
        // class is one of input, output, or inout
        // sub-classes are no_capture_core_signal, sample_only
        // no_sji_mux, no_sjo_mux, clock, dot6
        added_bscan_cell_options {list of class and sub-classes}
        output_during_force_disable 0 | 1 | X | Z
        auxiliary_output_pin pin_name
        auxiliary_output_enable_pin pin_name
        auxiliary_input_pin pin_name
        auxiliary_input_enable_pin pin_name
        ac_cell on | off
        bsdl_pin_name pin_name
        bonding_configurations {list of bonding options}
        enable_group_id ID
        extracted_pad_functions {list of functions and locations}
        output_trace_instance_list {list of pad instances}
        output_trace_mode_list {list of output direction modes}
        input_trace_instance_list {list of pad instances}
        input_trace_mode_list {list of input direction modes}
    }
}
boundary_scan_interface {
    force_disable          pin_name
    select_jtag_input      pin_name
    select_jtag_output     pin_name
    capture_shift_clock   pin_name
    capture_shift_clock_inv pin_name
    bscan_clock           pin_name
    update_clock          pin_name
    select                pin_name
    reset                 pin_name
    capture_en            pin_name
    shift_en              pin_name
    update_en              pin_name
    scan_in               pin_name
    scan_out               pin_name
    scan_out_launch_edge  posedge | negedge
    ac_init_clock0         pin_name
    ac_init_clock1         pin_name
```

```
    ac_signal           pin_name
    ac_mode_en         pin_name
}
logic_test_scan_chains {
    chain# {
        length          integer
        scan_in          port_pin_name
        scan_out         port_pin_name
        scan_in_capture_edge posedge | negedge
        scan_out_launch_edge posedge | negedge
    }
}
bscan_occ {
    present          on | off
    static_clock_control external
    capture_window_size integer
    interface {
        clock_sequence {pin_list}
        static_clock_control_mode pin_name
        capture_en       pin_name
        scan_en          pin_name
    }
    internal_nodes {
        clock_gate_output pin_name
    }
    parent_instance   instance_name
    instance_name     leaf_instance_name
}
```

The ports section gives you the information about the boundary scan cell associated to each port and where it gets connected on the pad cell. If a port was equipped with Auxiliary muxing logic as requested in the [AuxiliaryInputOutputPorts](#) wrapper, you can find the associated pin on the Logical grouping modules in the dictionary. The “boundary_scan_interface” section gives you the pins on the right hand side of the boundary scan interface module, illustrated in [Figure 10-34](#), for all pin functions needed by the boundary scan segments. Finally, the logic_test_scan_chains section is created if you used the max_segment_length_for_logictest property documented below. It describes all logictest chains created with their length, scan_in and scan out port and the edge timing for scan_in and scan_out with respect to the shift clock.

See the example below on how to retrieve the dictionary and manipulate the chain data.

Note

 The mini OCC is added to the BoundaryScan interface when max_segment_length_for_logictest is an integer or unlimited. For more information about the mini OCC, refer to the [Sib](#) command description.

Arguments

- `ijtag_host_node : id ;`

A required property that specifies the ScanInterface that is hosting the boundary scan register. The format of the *id* is either “Tap(*id*)” or “HostScanInterface(*id*)”. You use Tap(*id*) when the boundary scan host is a Tap controller specified in the IJtagNetwork wrapper of the current DftSpecification wrapper. You use HostScanInterface(*id*) when the host of the boundary scan is a TAP that already exists in the design. The IJtagNetwork must have a HostScanInterface(*id*) wrapper with `design_instance` and `scan_interface` properties that identify the TAP and the bscan host interface as shown in “[Example 2](#)” on page 3246 in the [HostScanInterface/Interface](#) section.

- `pin_order_file : file_name ;`

An optional property that specifies the name of a file that provides an ordered list of ports that the tool will use to order the boundary-scan cells. The *file_name* can be an absolute or relative path. If the path is relative, it is taken with respect to the current working directory.

The pin order file format is shown in [Figure 10-37](#) on page 3396. It consists of a four-column table in which the boundary scan ordering is taken from top to bottom. If you do not have a pin order file, you will not be able to specify more than one LogicalGroup wrapper in the [LogicalGroups](#) wrapper. If you have read in a DEF file using the `read_def` command, the order will be extracted from the placement of the pad cell in order to minimize the routing of the boundary scan chain.

The first column lists the name of the port as it exists in the chip netlist.

The second column lists the pin name the port maps to in its package. This information is only forwarded to the BSDL file and only needs to be accurate by the time the BSDL is used in the board environment. If you specify a “-” or nothing in this column, an integer will be automatically assigned to this port in the BSDL file which is often good enough given that chips today have multiple bounding configurations and packages. The pin name typically needs to be adjusted long after the design has been completed. You can specify nothing in the second column if you do not have anything to specify in the third or forth column. You must use “-” if you want to let the tool pick an integer for the pin name but you have entries to specify in the third or forth columns.

The third column is used to attach options to specific ports. It is equivalent to using the `port_name_pattern` property inside the [BoundaryScanCellOptions](#) wrapper. When specifying more than one option for a given port, you separate the options with a comma. If you have no option, you can specify “-”. You can also specify nothing at all in the third column as long as you do not have an entry in the LogicalGroups column.

The fourth column is used to group the boundary-scan cells into logical modules. It is equivalent to using the [LogicalGroups](#) wrapper inside the BoundaryScan wrapper. A “-” or no value at all in the forth column means that the given port belongs to the logical grouping of the port above it.

You can simply supply a `pin_order_file` which lists the ordered port list and nothing else in the second, third, and forth column. The pin names will be automatically assigned to

integers which you can adjust later on when the true pin names on the package are known. The options and the logical grouping can be specified more easily in the BoundaryScan wrapper because wildcards are supported.

Figure 10-37. Example pin_order_file

// PortName	PinName	OptionList	LogicalGroups
// -----	-----	-----	-----
IN1	1	-	left_0
IN2	2	-	-
GND1 [2]	3	GND	-
GND1 [1]	3A	GND	left_1
GND1 [0]	X1	GND	-
IN3	4	-	-
IN4	5	-	top
OUT1	6	-	-
OUT2	7	-	right_0
VDD1	Y1	PWR	-
TN	9	CE1	bottom
SCANIN	10	NJTAG	-
NC1	-	NC	right_0
TCKP	11	TCK	right_0
TMSP	12	TMS	right_0
TRSTP	13	TRST	right_0
TDIP	15	TDI	right_0
TDOP	16	TDO	right_0

- `interface_parent_instance : parent_instance ;`

An optional property that identifies the design module in which the tool will place the boundary scan interface. The default placement will be at the top level if this property is not specified.

- `outputs_per_enable_cell : int ;`

An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. If the enable pin of the output pad cell is tied on or off, the tool is free to associate any output pad cell to any given enable cell in order to meet the outputs_per_enable_cell constraints. If, on the other hand, a group of output pad cells have their enable pin controlled by a single functional net, the tool inserts N enable boundary-scan cells and automatically separates the fanout of the enable signal such that the fanout of each enable cell is smaller than or equal to the specified outputs_per_enable_cell value. The splitting of the functional enable fanout is illustrated in the example of the [LogicalGroups](#) wrapper in [Figure 10-25](#) and [Figure 10-26](#).

- `max_segment_length_for_logictest : int | unlimited | off ;`

An optional property that when set to “unlimited” or an integer value, instructs the tool to segment the boundary scan chains into smaller segments to be reused by scan chain testing, EDT, or LogicBIST. If “unlimited” is specified the boundary scan chain will be accessible as one long scan chain.

Note

 To use the boundary scan cells to isolate the core from the primary input ports, add the int_ltest_en DFT signal using the [add_dft_signals](#) command. The signal will be connected to the boundary scan interface module and force the to_select_jtag_input port to 1 when asserted.

If you have inout ports or if you want to save power during the LogicBIST modes, add the output_pad_disable DFT signal to allow turning off the tri-state output buffers. The signal will be connected to the boundary scan interface module and force the to_force_disable port to one when asserted. The output pads used for channel outputs will be turned on during shift and tri-stated in capture mode. This will avoid bus conflict during ATPG and save burning power in the output pad buffers. The pad buffers will be tested by the boundary scan test patterns.

If an integer is specified, the boundary scan chain will be partitioned into segments of the specified size. Note that segments can end up having a different length because Tessent Shell is not able split the boundary scan chain at some positions, such as within a given [EmbeddedBoundaryScan](#) segment. You should choose a number that does not exceed the length of the other scan chains in the design because the boundary scan segments will otherwise increase your test time.

The default value of “off” indicates that the boundary scan chain is not reused during logic test.

- tristate_enable_non_contacted_test_support : on | off ;

An optional property that when set to “on” generates the hardware needed to test if the pad driver of a bidirectional non-contacted pad can be disabled by the global force_disable signal and the local output enable signal. Pattern generation is controlled through specifying the tristate_enable_non_contacted test for [RunTest](#).

The generated hardware is controlled by the two global DFT control signals tristate_enable_nc_test and tristate_enable_nc_test_type, which can be shown with the [report_dft_signal_names](#) command. It is recommended to specify these control signals with [add_dft_control_points](#) before creating the DFT specification so the IJTAG network will have TDRs to control the generated hardware.

This feature is not available in the embedded boundary scan implementation.

- tck_period : *period* ;

An optional property that instructs the tool to use a different default period of the test clock. The default value of the Tessent BoundaryScan period is 100ns.

Examples

The following example defines a BoundaryScan wrapper. The ijttag_host_node property is used to specify that the boundary scan chain is to be connected to a Tap controller defined in the IjttagNetwork wrapper. The file *./mychip.pin_order* is referenced using the pin_order_file property to order the boundary-scan cell so as to minimize routing. See the [set_boundary_scan_port_options -pin_order_file](#) option to have the [create_dft_specification](#)

command create a DftSpecification that points to an existing pin order file. See the [read_def](#) command to learn how to load a def file and have the pin order file automatically created following the placement of the ports in your design.

In this example, the pin order file only contains the column that lists the port names. The PinName column is omitted and the tool will auto assign integers to the pin names in the BSDL file. The Option column is also not used and, instead, the Options are specified using the [BoundaryScanCellOptions](#) wrapper. The outputs_per_enable_cell property is specified with a value of 8 which means that one enable boundary-scan cell will be inserted for every eight ports. A group of 22 pads in this design share a common functional enable signal. The tool will insert 3 enable cells for this group, each controlling the enable pin of 7 or 8 output pad buffers. The max_segment_length_for_logictest property is used to specify that the tool is to provide access to the boundary scan chain for logictest in chain segments no longer than 200 flip-flops long.

```
DftSpecification(mychip,rtl) {
    BoundaryScan {
        ijtag_host_node : Tap(main) ;
        pin_order_file : ./mychip.pin_order ;
        outputs_per_enable_cell : 8 ;
        max_segment_length_for_logictest : 200 ;
    }
}
```

Because the above example used the “max_segment_length_for_logictest” property, the instrument dictionary “mentor::jtag_bscan::DftSpecification” contains the “logic_test_scan_chains” section which can be accessed this way:

```
set logic_test_enable [get_instr_dict \
    mentor::jtag_bscan::DftSpecification logic_test_scan_chains]
foreach chain [dict keys $logic_test_enable chain*] {
    dict with logic_test_enable $chain {
        puts "length = $length"
        puts "scan_in = $scan_in"
        puts "scan_out = $scan_out"
        puts "scan_in_capture_edge = $scan_in_capture_edge"
        puts "scan_out_launch_edge = $scan_out_launch_edge"
    }
}
```

The dictionary can also be exported to file and re-used in a separate tool invocation when using the -variable_name switch and the redirection option “>”. The generated file can then later be sourced into a dofile to recover a variable called “bscan_ltest_chains” which holds the value of the dictionary.

```
format_dictionary [
    get_instrument_dictionary mentor::jtag_bscan::DftSpecification \
        logic_test_scan_chains \
] -variable_name bscan_ltest_chains > bscan_ltest_chains.dict
```

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        ImplementationOptions {
            clocking           : tck | gated_tck | gated_tck_inv ; // *DefSpec
            update_stage       : flop | latch ;
            scan_path_retiming : flop | latch ;
        }
    }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- **clocking: tck | gated_tck | gated_tck_inv ;**

A property that specifies how the clocking of the boundary-scan cell should be constructed. When specifying tck, the boundary-scan cells are clocked with the free-running TCK clock and the capture and shift is enabled using the capture_en and shift_en functions. This option requires one extra multiplexer per boundary-scan cell as compared to the gated_tck or gated_tck_inv implementations. When specifying gated_tck, a gated low version of TCK is used and the gating lets the TCK go through during the capture and shift enable pulse. When specifying gated_tck_inv, a gated high version of TCK is used and the gating lets the inverted TCK go through during the capture and shift enable pulse. All three clocking specifications behaves identically in IEEE 1149.1 mode. Using TCK versus the other two gated TCK options is a trade-off between area and the number and sizes of clock trees. Note that in all three cases, the boundary-scan cells are always built such that the scanin of each cell is strobed on the rising edge of TCK and the scanout is launched off the negedge of TCK to make sure boundary-scan cells will always work without having to balance TCK to all boundary scan cells which makes the ground bounce issue worst.

The gated_tck_inv option is a legacy clocking mode and is the one that is compatible with the existing LogicVision burst mode architecture. For more information, see the “[BurstMode Logic BIST Architecture](#)” section in the *LV Flow User’s Manual*.

[Table 10-3](#) illustrates the boundary scan cell implementations based on the clocking, update_stage and scan_path_retiming values.

- **update_stage: flop | latch ;**

A property that specifies if the update stage is to be implemented using a latch or a flop.

[Table 10-3](#) illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values.

- scan_path_retiming : flop | latch ;

A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

Table 10-3 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values. Refer to **Table 10-5** to see the implementation of an input and sample_only boundary scan cell using the default implementation options. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

Table 10-7. Illustration of Different Boundary Scan Implementations

- 1) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck, and update_stage and scan_path_retiming are both set to latch.

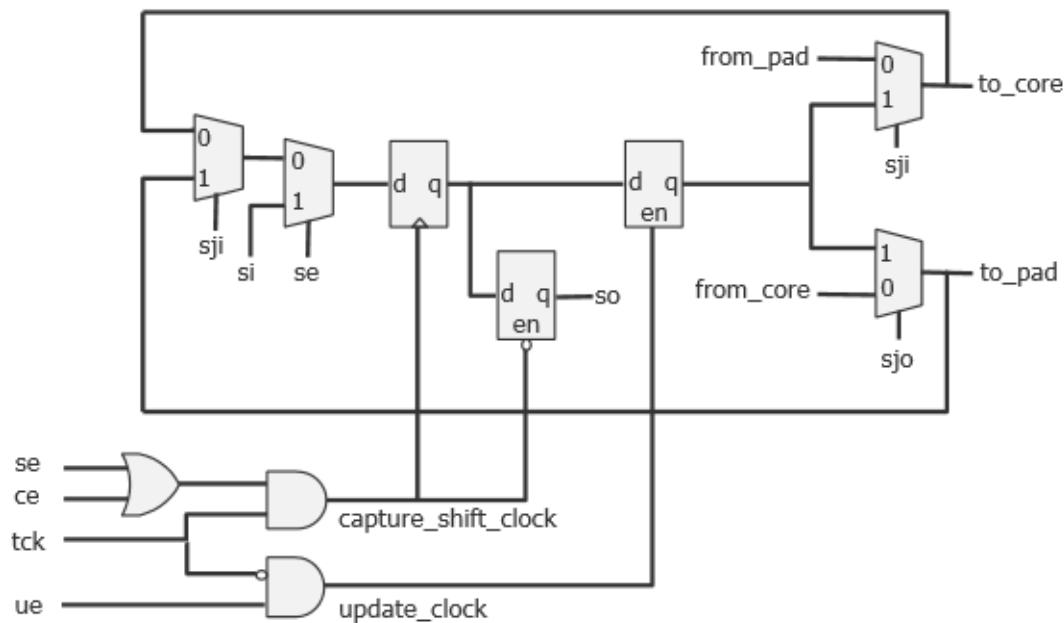


Table 10-7. Illustration of Different Boundary Scan Implementations (cont.)

2) This example shows how the boundary-scan cells are constructed when the clocking is set to gated_tck and update_stage and scan_path_retimng are both set to flop. Compared to the prior example, this shows how changing update_stage and scan_path_retimng from latch to flop simply converts the latch into a flop.

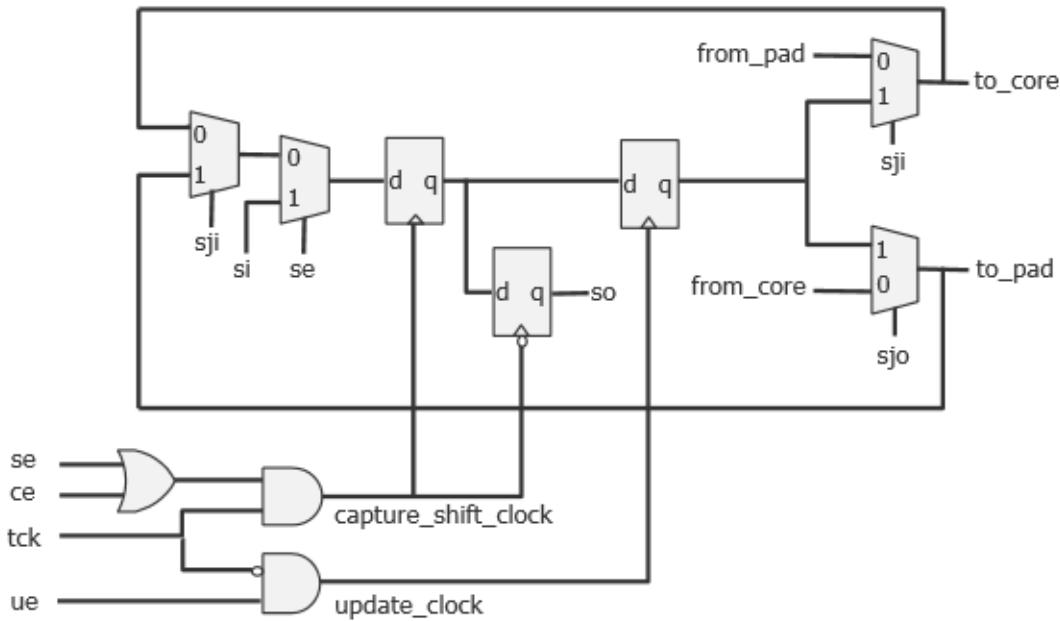
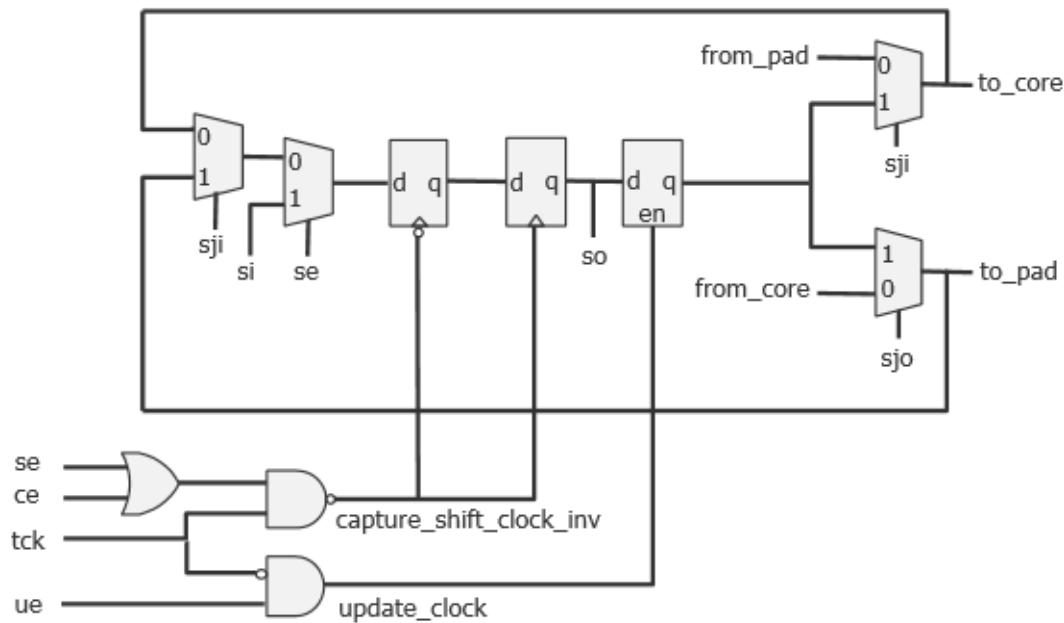
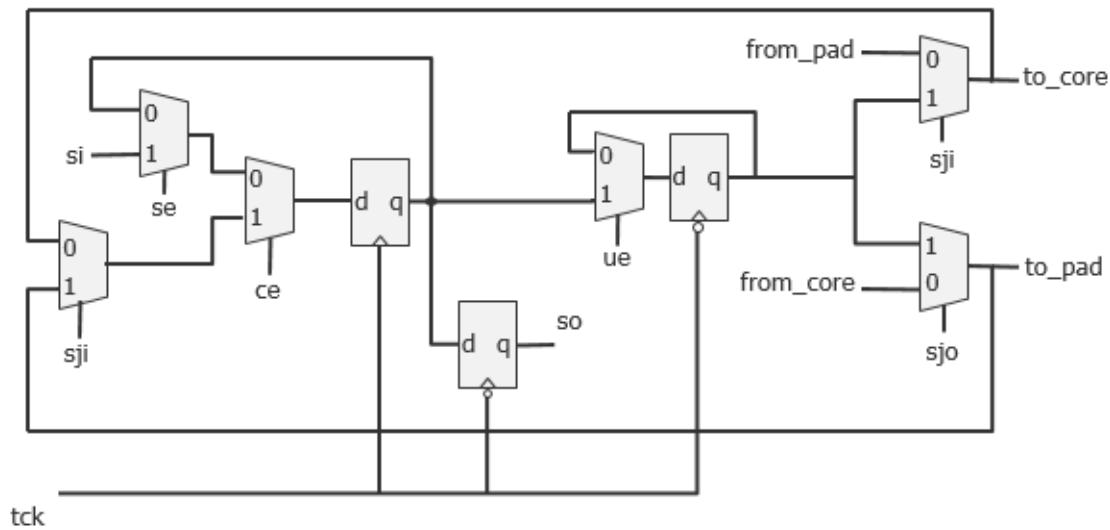


Table 10-7. Illustration of Different Boundary Scan Implementations (cont.)

3) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck_inv, and update_stage and scan_path_retimining are both set to latch. This is a legacy mode that makes the boundary-scan cell compatible with the LogicVision burst mode architecture. Notice how the implementation for gated_tck and gated_tck_inv are almost the same; only the active-low retiming flop is placed before the data flop.



4) This example shows how the boundary-scan cells are constructed when clocking is set to tck, and both update_stage and scan_path_retimining are set to flop. Notice the two extra multiplexers that are needed in each cell.



Examples

The following example sets the clocking property to gated_tck_inv and the update_stage to latch. The scan_path_retiming is unspecified so it defaults to latch.

```
DftSpecification(mychip,rtl) {
    BoundaryScan {
        ImplementationOptions {
            clocking      : gate_tck_inv ;
            update_stage : latch ;
        }
    }
}
```

BoundaryScanCellOptions

Assigns options to boundary scan ports.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        BoundaryScanCellOptions {
            port_name_pattern : option, ... ; // repeatable
        }
    }
}
```

Description

This wrapper is used to assign options to boundary scan ports. This wrapper replaces the Overrides wrapper in the .etassemble file.

Arguments

- *port_pattern_name* : option, ...;

A syntax used to assign one or more option to a set of ports matched by the specified *port_pattern_name*. The *port_pattern_name* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match “ABC00”, “ABC01”, and “ABC02”.

[Table 10-4](#) lists the option names and their meanings. The option can be specified in the BoundaryScanCellOptions wrapper or in the third column of the pin_order_file, as explained in the section.

Table 10-8. BoundaryScanCellOptions list

Option name	Meaning
add_dot6_from_pad_cell	Used to identify AC input and inout ports where the from_pad data signal will be intercepted with an additional boundary scan cell. The boundary scan cells sampling the AC pad Test Receivers will be added as normally done.
analog	Used to identify input, output, and inout ports that are analog. These pins do not have a boundary scan cell and are reported as linkage pins in the BSDL file.

Table 10-8. BoundaryScanCellOptions list (cont.)

Option name	Meaning
compliance_enable0	<p>Used to identify input and inout ports that are active low compliance enable pins. These ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven low for the chip to behave in accordance with the BSDL description. The compliance enable zero pins are automatically asserted low in any pattern that makes use of the TAP controller.</p> <p>This option is automatically set when you use the <code>-enable_low_compliance_enables</code> switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the <code>-enable_low_compliance_enables</code> switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.</p>
compliance_enable1	<p>Used to identify input and inout ports that are active high compliance enable pins. Those ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven high for the chip to behave in accordance with the BSDL description. The compliance enable one ports are automatically asserted high in any pattern that makes use of the TAP controller.</p> <p>This option is automatically set when you use the <code>-enable_high_compliance_enables</code> switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the <code>-enable_high_compliance_enables</code> switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.</p>
clock	<p>Used to identify input, output, and inout ports that are clocks. Specifying this value alters the RTL and the BSDL definition of the generated boundary-scan cells associated with the port.</p> <ul style="list-style-type: none"> • If the port is of type input or inout, the boundary-scan cell will have a BSDL function of “clock” and no JTAG multiplexer is inserted along the functional input path just as if the <code>sample_only</code> option was specified. • If the port is of type output or inout, the boundary-scan cell is modified to capture itself instead of the <code>from_core</code> signal in the output direction just as if the <code>no_capture_core_signal</code> option was specified.

Table 10-8. BoundaryScanCellOptions list (cont.)

Option name	Meaning
dont_touch	Used to identify input, output, and inout ports that are to be fully excluded from boundary scan. It is similar to using the no_bscan_cell option except that for output and inout ports, the output enable pin of the output pad buffer is not intercepted with the force_disable signal such that the port will not go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
ground	Used to identify input and inout ports that are ground pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the ground option may or may not exist in the design. If they do not exist in the design file, the port name pattern is not allowed to include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to ground prior to running the check_design_rules command are automatically identified as ground pins in the created DftSpecification wrapper when you run the create_dft_specification command.
input_only	Used to identify inout ports that are to be treated as input-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file will document the port as input, and the port will only be equipped with an input-type boundary-scan cell.
no_bscan_cell	Used to identify input, output, and inout ports that are not to be equipped with a boundary-scan cell. It is similar to using the dont_touch option except that for output and inout ports, the output enable pin of the output pad buffer is still intercepted with the force_disable signal such that the port will still go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
no_connect	Used to identify input, output, and inout ports that are not connected to package pins. Those ports are treated as dont_touch pins and they are not reported at all in the BSDL file.

Table 10-8. BoundaryScanCellOptions list (cont.)

Option name	Meaning
no_capture_core_signal	Used to identify output and inout ports that are to be equipped with boundary-scan cells that do not capture the functional core signal but instead hold their values during the capture cycle of the INTEST instruction. This option is useful when you are reusing the boundary scan chain as isolation during logictest and you want to avoid the boundary-scan cells from capturing unknowns when the from_core signal is not controllable during logictest. Avoiding unknowns is a requirement if you are using LogicBIST.
output_only	Used to identify inout ports that are to be treated as output-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file will document the port as output, and the port will only be equipped with an output-type boundary-scan cell.
power	Used to identify input and inout ports that are power pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the power option may or may not exist in the design. If they do not exist in the design file, the port name pattern is not allowed to include the asterisk wildcard symbol “*” but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to power prior to running the check_design_rules command are automatically identified as power pins in the created DftSpecification wrapper when you run the create_dft_specification command.
sample_only	Used to identify input and inout ports that are not to have a JTAG multiplexer inserted along the functional input path. The fanout of these ports cannot be intercepted by the boundary-scan cell value during the INTEST instruction. Also, if you reuse the boundary scan as isolation during logictest, these ports will not be isolated. This option is inferred for input or inout ports having the clock option. You typically also want to use it for Async reset input ports.

The differences between a normal input boundary scan cell and one assigned the sample_only option are illustrated in [Table 10-5](#) below. Refer to [Table 10-3](#) for information

on how an inout cell is constructed. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

Table 10-9. Illustration of the “sample_only” Boundary Scan Cell Option

1) This example shows a normal input boundary scan cell. The multiplexer shown on the right intercepts the from_pad connection of the pad cell. The boundary scan cell can drive a value to the core in INTEST mode.

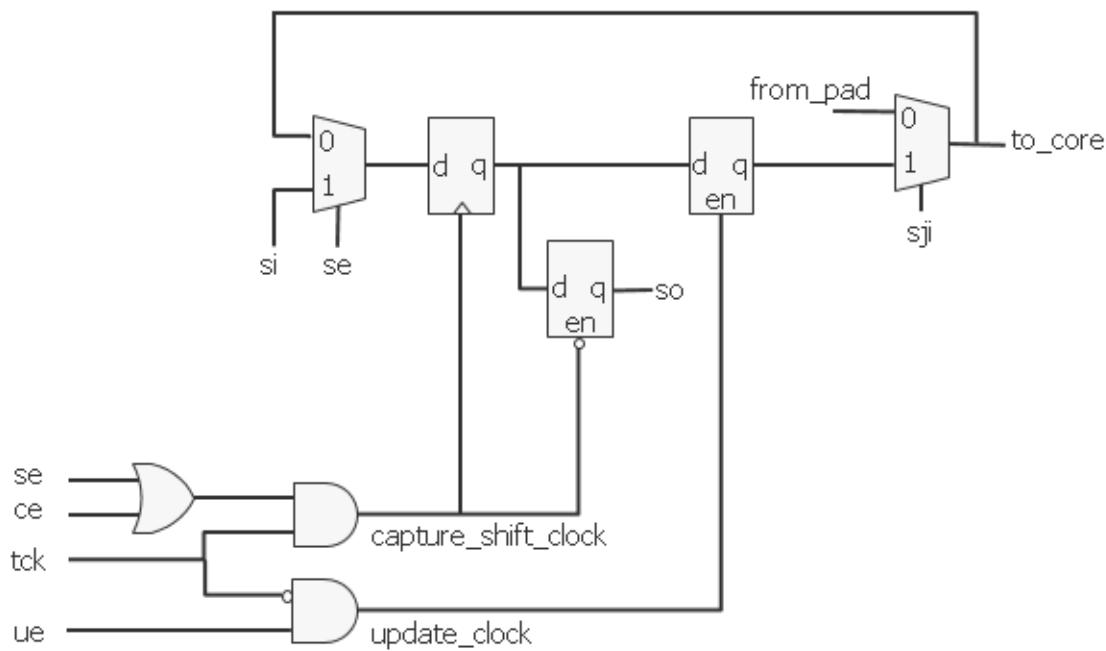
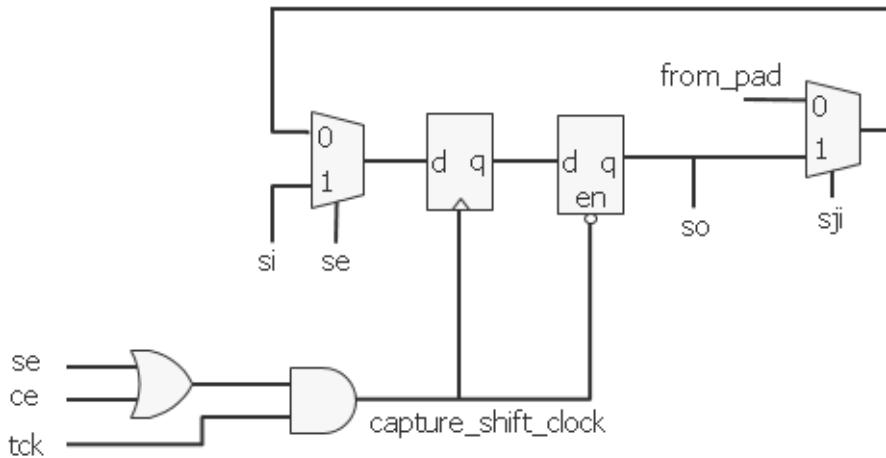


Table 10-9. Illustration of the “sample_only” Boundary Scan Cell Option (cont.)

2) The example below shows a sample_only input boundary scan cell. The multiplexer shown on the right will no longer intercept the from_pad connection of the pad cell, but will simply connect to it, preserving the direct connection between the pad and core. Note that compared to the normal input boundary scan cell, the update latch has been removed and the retiming latch is now connected to the multiplexer on the right.



Examples

The following example uses the `BoundaryScanCellOptions` wrapper to identify VDD and VSS ports in the BSDL file. These port do not exist in the design. The pattern `clk*` is used to match any ports in the design starting with the string “clk” as clock ports. They will not have a JTAG mux inserted along their functional input path, and they will be identified as Clock ports in the BSDL file.

```
DftSpecification(mychip,rtl) {
    BoundaryScan {
        BoundaryScanCellOptions {
            VDD%d[0:32] : power ;
            VSS%d[0:56] : ground ;
            clk* : clock;
        }
    }
}
```

LogicalGroups

Specifies how to group the boundary-scan cells into logical grouping modules.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        LogicalGroups {
            LogicalGroup(group_name) {
                first_port : port_name ;
                parent_instance : parent_instance ;
                leaf_instance_name : leaf_instance_name ;
            }
        }
    }
}
```

Description

A wrapper that specifies how to group the boundary-scan cells into logical grouping modules. You can specify more than one LogicalGroup wrapper only when you have specified the pin_order_file property in the [BoundaryScan](#) wrapper.

You use the first_port property to identify the first port in the pin order file that belongs to the logical grouping. The association between ports and the LogicalGroup can also be specified in the forth column of the pin order file. If you do not specify any LogicalGroup(*group_name*) wrapper, all boundary scan cells are grouped into a single module, the module is instantiated into the top module, and the leaf instance name is as follows:

design_name_design_id_tessent_bscan_logical_group_DEF_inst

as if you had specified an empty LogicalGroup(DEF) wrapper.

You use the parent_instance property to specify where you want the logical grouping module to be instantiated. By default, it will be instantiated in the top module. You use the leaf_instance_name property to specify the leaf instance name used when instantiating the module if you are not satisfied with the default value. The default leaf_instance_name used is *design_name_design_id_tessent_bscan_logical_group_group_name_inst*.

The example below shows how to specify the parent_instance property in order to insert each logical grouping module in the same instance where the pins or nets to be intercepted by the boundary-scan cells are located.

Arguments

- *group_name*

A string that uniquely identifies each logical group. The string starts with a letter and it is allowed to be followed by any number of letters, numbers, and underscores. The group_name is used in constructing the grouping module name as follows:

design_name_design_id_tessent_bscan_logical_group_group_name

- `first_port : port_name ;`

A property that specifies a port name that exists in the specified pin_order_file. This property is optional when there is only one LogicalGroup wrapper but it is required when there are more than one. Specifying a port_name for the first_port property is equivalent to specifying the group_name in the forth column of that port in the pin_order_file. See the pin_order_file property description in the BoundaryScan wrapper for a description of that file format and usage.

- `parent_instance : parent_instance ;`

A property that controls where the logical grouping module will be instantiated. By default, it is instantiated in the root module but you can push it down into instances. See the following example to understand the requirement that must be met when choosing the parent_instance of the logical grouping module.

If the specified parent_instance is not unique, it will be uniquified before the logical grouping module is inserted in it. You can do bottom up insertion using the [EmbeddedBoundaryScan](#) wrapper on sub modules to insert segments of boundary scan chains into repeated modules, and then reuse them as building blocks at the next level up if you want to avoid uniquifying the parent_instance. Specify the “[set_design_level sub_block](#)” command and option when inserting boundary scan into those sub modules.

- `leaf_instance_name : leaf_instance_name ;`

A property that specifies a different leaf name when instantiating the logical grouping module into its parent instance. The default value when unspecified is `design_name_design_id_tessent_bscan_logical_group_group_name_inst`.

If you choose to change it, you do not have to worry about picking a unique leaf name in the given parent_instance. The tool will automatically append the `instance_uniquification_suffix` to your specified leaf instance name in case of conflicts. See the [set_insertion_options](#) command description for a complete description of the `-instance_unification_suffix` option.

Examples

The following example illustrates the use of the parent_instance property of the LogicalGroup wrapper to meet the rule that the logical grouping module must be instantiated where all the intercepted nets are visible. [Figure 10-22](#) shows the circuit before the insertion of the boundary-scan cells. You can see that the functional connections to and from the pad cells associated with ports P3 and P4 do not exist outside the B1_I instance. In this example, a single logical grouping module is specified so it can only be instantiated inside B1_I as illustrated in [Figure 10-23](#).

```
BoundaryScan {
    LogicalGroups {
        LogicalGroup(g1) {
            parent_instance : B1_I ;
        }
    }
}
```

The same example is modified to use a pin_order file thus enabling the splitting of the boundary-scan cells into more than one logical grouping module. The pin order file defines the order from P1 to P4. Two logical grouping modules are specified in which the first one, g1, starts with P1 and goes until P2 and the second one, g2, starts with P3 and goes until P4. The g2 group is still restricted to having its parent_instance property as B1_I because this is the only place where all of the connections to be intercepted are visible. The g1 group is, on the other hand, allowed to be instantiated in four locations because all of the connections to be intercepted are visible in those four locations. The parent_instance is shown for the four options and the corresponding locations of the logical grouping modules are shown in [Figure 10-24](#).

```
BoundaryScan {
    pin_order_file      : mychip.pin_order
    LogicalGroups {
        LogicalGroup(g1) {
            first_port      : P1 ;
            parent_instance : B1_I ;           // Option1
            parent_instance : . ;           // Option2
            parent_instance : B2_I ;           // Option3
            parent_instance : B2_I/B3_I ;     // Option4
        }
        LogicalGroup(g2) {
            first_port      : P3 ;
            parent_instance : B1_I ;
        }
    }
}
```

The example also illustrates that you can specify the outputs_per_enable_cell property with a value smaller than the actual fanout count of the functional enable signal. In this case, the functional enable signal fans outs to two output pads. One enable boundary-scan cell is inserted per output port, requiring the tool to split the output enable connectivity to each pad.

[Figure 10-25](#) shows this example with option1 for the parent_instance property. No new ports need to be created. The functional enable signal is observable by two enable bscan cells, and one connection per output pad is made between the logical grouping module and the enable pin of the output pad cells.

[Figure 10-26](#) shows this example with option3 for the parent_instance property. In this case, the connections between the logical grouping module and the output pads require the creation of new pins as illustrated with the orange net crossing the hierarchy. If you are familiar with Tessent Boundary Scan within ETAssemble, the flexibility to precisely control the fanout of the enable signals was not present in that tool. All of those existing limitations are removed within Tessent Shell, and a net driven by RTL can now be intercepted and split for boundary scan usage. In ETAssemble, you needed the functional enable signal to be identified on a pin which often meant you had to insert buffers in the RTL to enable the Boundary Scan to be placed in the same module as the pads.

```
BoundaryScan {
    pin_order_file      : mychip.pin_order
    OutputPerEnableCell : 1 ;
    LogicalGroups {
        LogicalGroup(g1) {
            first_port      : P1 ;
            parent_instance : B1_I ;           // Option1
            parent_instance : B2_I ;           // Option3
        }
        LogicalGroup(g2) {
            first_port      : P3 ;
            parent_instance : B1_I ;
        }
    }
}
```

Figure 10-38. Example Design With Pads

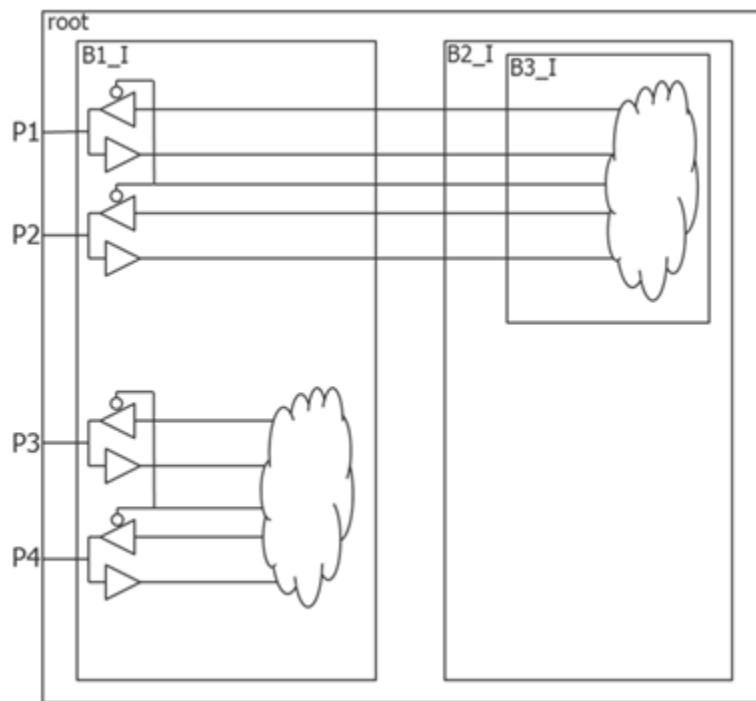


Figure 10-39. Single Logical Group Example

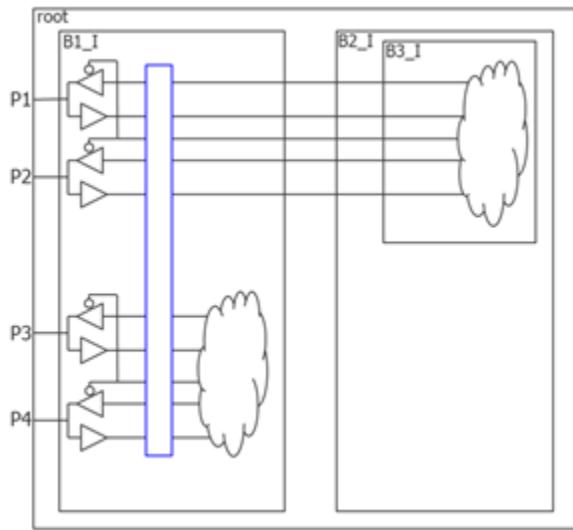


Figure 10-40. Two Logical Groups Example

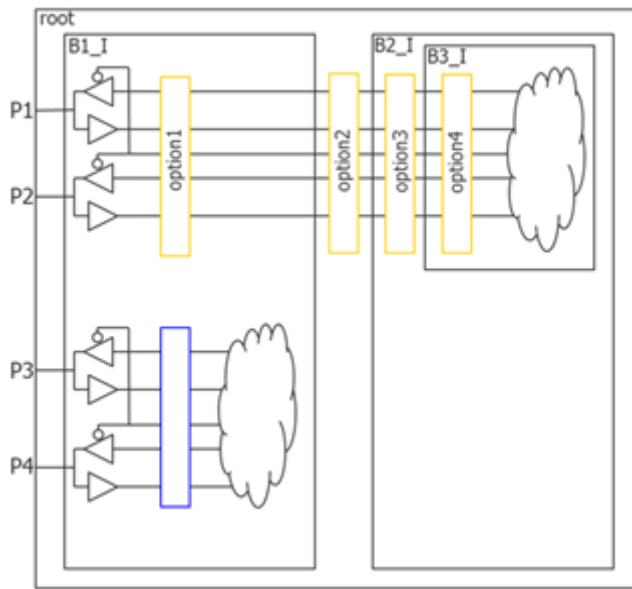


Figure 10-41. Two Logical Groups Example1 with One Output per Enable Cells

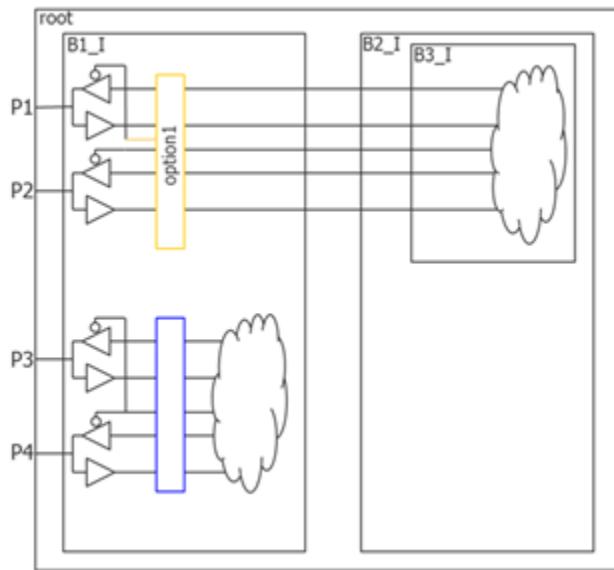
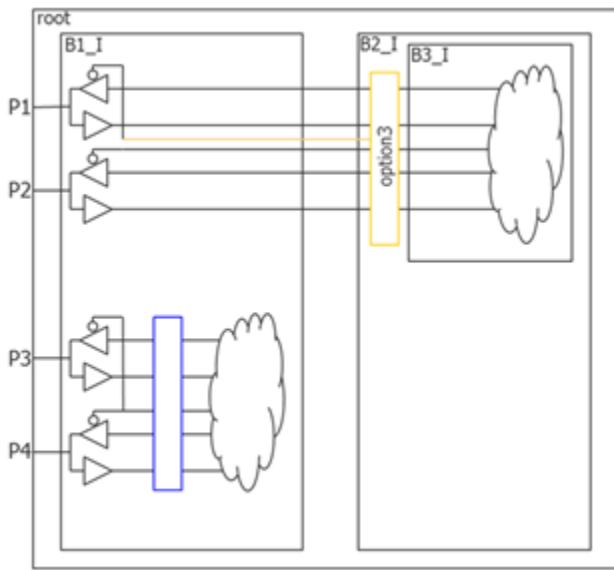


Figure 10-42. Two Logical Groups Example2 with One Output per Enable Cells



ACModeOptions

Specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        ACModeOptions {
            test_receiver_init_clock_value : initialization_event ;
                // Legal: logic_low logic_high
                // falling_edge rising_edge auto
            pulse_min_duration : time | auto ;
            train_minimum_count : count | unused ;
            train_max_duration : time | unlimited ;
            ACGroup(group_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                port_list : port_name, ... ;
            }
        }
    }
    EmbeddedBoundaryScan {
        ACModeOptions {
            test_receiver_init_clock_value : initialization_event ;
                // Legal: logic_low logic_high
                // falling_edge rising_edge auto
            ACGroup(group_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                port_list : port_name, ... ;
            }
        }
    }
}
```

Description

A wrapper that specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain. You use this wrapper to define AC enable cells and associate them with a group of ports. You also use this wrapper to specify the event type that initializes the test receiver when the value is not described in the cell library. Finally, you can use this wrapper to specify three properties that are simply forwarded into the BSDL file.

Arguments

- `test_receiver_init_clock_value : initialization_event ;`

A property that is useful when the cell library description of the dot6 pad cell uses the legacy `pad_init_clock_dot6` attribute instead of one of the new attributes that precisely describe the initialization event: `pad_init_posedge_clock_dot6`, `pad_init_negedge_clock_dot6`, `pad_init_enable_high_dot6`, and `pad_init_enable_low_dot6`. If the value is “auto” and the dot6 pad cell uses the legacy `pad_init_clock_dot6` attribute instead of one of the four new attributes, the tool issues an error; you are required to specify the initialization event when it cannot be inferred from the cell library.

- `pulse_min_duration : time | auto ;`

An optional property that defines the minimum time to use between the rising and a falling edge during the AC tests. This property is forwarded into the BSDL file using the AIO_EXTEST_Pulse_Execution attribute. When unspecified or specified to “auto”, the minimum pulse duration is chosen so as not to be smaller than three times the larger of the HP_time and LP_Time BSDL values of each pad cell.

```
attribute AIO_EXTEST_Pulse_Execution of <current_design>: entity is
    "Wait_Duration <pulse_min_duration_in_seconds>" ;
```

- `train_minimum_count : count | unused ;`

An optional property that defines the minimum number of pulses the train test must have. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. The property defaults to “unused” when unspecified. Specifying a value of one effectively makes the train test equivalent to the pulse test.

```
attribute AIO_EXTEST_Train_Execution of <current_design>: entity is
    "train <train_minimum_count>, maximum_time <train_max_duration>" ;
```

- `train_max_duration : time | unlimited ;`

An optional property that defines a maximum time the train test must take. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. It defaults to “unlimited” when unspecified.

- `group_name`

A string that uniquely identifies the ACGroup. This name is not reflected in the BSDL file but is used to name the leaf instance name of the AC group enable cell in the logical grouping module.

If no AcGroup wrappers are specified, then all AC ports are active during the AC tests. If you specified at least one ACGroup wrapper but you have some AC ports not associated with an ACGroup, they will always be active during the AC tests. In this situation, a warning is generated to let you know that not all AC pins were equipped with an AC enable cell. You can define an extra ACGroup wrapper without a port_list property to indicate that you want all AC ports not associated with another ACGroup wrapper to be associated with that one.

- `insert_before_port : port_name ;`

A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the [BoundaryScan](#) wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

- `insert_after_port : port_name ;`

A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the `pin_order_file` property inside the [BoundaryScan](#) wrapper is specified. The specification of the `insert_before_port` property is mutually exclusive with the specification of the `insert_after_port` property. If neither the `insert_before_port` nor `insert_after_port` properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the `AIO_Pin_Behavior` attribute.

- `port_list : port_name_pattern, ... ;`

A property used to list the port names that belong to the AC group. The `port_name_pattern` can include the asterisk “*” as a wildcard and `%#d[lb:rb]` as a counter. The `#` entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, `ABC%2d[0:2]` will match ABC00, ABC01, and ABC02.

You are allowed to have one `ACGroup` wrapper with an unspecified `port_list`. This `ACGroup` will collect all AC ports that are not already associated with another `ACGroup` wrapper.

Examples

The following example defines an `ACGroup` wrapper with the name “`pci`” to specify that all ports starting with `PCI_` belong to the group. All other AC ports are associated with the “`other`” group. The “`other`” `ACGroup` collects all other AC ports because it does not specify the `port_list` property.

```
DftSpecification(my_chip,rtl) {
    BoundaryScan {
        AcGroups {
            ACGroup(pci) {
                port_list : PCI_* ;
            }
            ACGroup(other) {
            }
        }
    }
}
```

AuxiliaryInputOutputPorts

Specifies two lists of port name patterns to be used as Auxiliary input and/or output ports during some test modes.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        AuxiliaryInputOutputPorts {
            auxiliary_input_ports : port_name_pattern, ... ;
            auxiliary_output_ports : port_name_pattern, ... ;
        }
    }
    EmbeddedBoundaryScan{
        AuxiliaryInputOutputPorts {
            internal_auxiliary_input_ports : port_name_pattern, ... ;
            internal_auxiliary_output_ports : port_name_pattern, ... ;
            external_auxiliary_input_ports : port_name_pattern, ... ;
            external_auxiliary_output_ports : port_name_pattern, ... ;
        }
    }
}
```

Description

A wrapper that specifies two lists of port name patterns to be used as Auxiliary input and/or output ports during some test modes. An inout port is allowed to be specified in both lists at the same time. In the output directions, a 2-to-1 multiplexer (shown in green in [Figure 10-29](#)) is added on the test side of the JTAG multiplexer (shown in red in [Figure 10-29](#)). The auxiliary output multiplexing is added in the boundary-scan cell to avoid cascading two multiplexers along the functional output path. For auxiliary input logic, an AND gate is used to prevent the aux_in_data signal from toggling in functional mode. The aux_in_en signal is also used to force the output pad driver off when adding AuxIn logic on an inout pad.

The aux_out_enable and aux_in_enable signals are kept separate for each port to provide full flexibility as to which output and input ports are enabled to carry input or output auxiliary data in any given test mode. The ports equipped with auxiliary data input and output logic are useful for supplying channel input and output data to EDT controllers. They are also useful to provide the source of the scan_en, test_clock, and edt_update DFT signals. See the description of the [add_dft_signals](#) command for more info. Notice that the aux_out_en and aux_in_en signals have higher priority than the select_jtag_input and select_jtag_output signals. The reason for this is explained below but has the implication that the aux_out_en and aux_in_en sources must reset off and remain off when using the IEEE 1149.1 test mode of the chip. This requirement is met if you use the DFT signals created by the [add_dft_signals](#) to control those pins.

[Figure 10-27](#) shows the typical boundary scan cell inserted on a inout pad. The red multiplexers are those referred to as JTAG multiplexers in the first paragraph. They are controlled by the select_jtag_output to allow the output data and the output enable values to be controlled by the Boundary Scan cells in IEEE 1149.1 test modes.

Figure 10-43. Typical Boundary Scan Cell Inserted for an Inout Pad

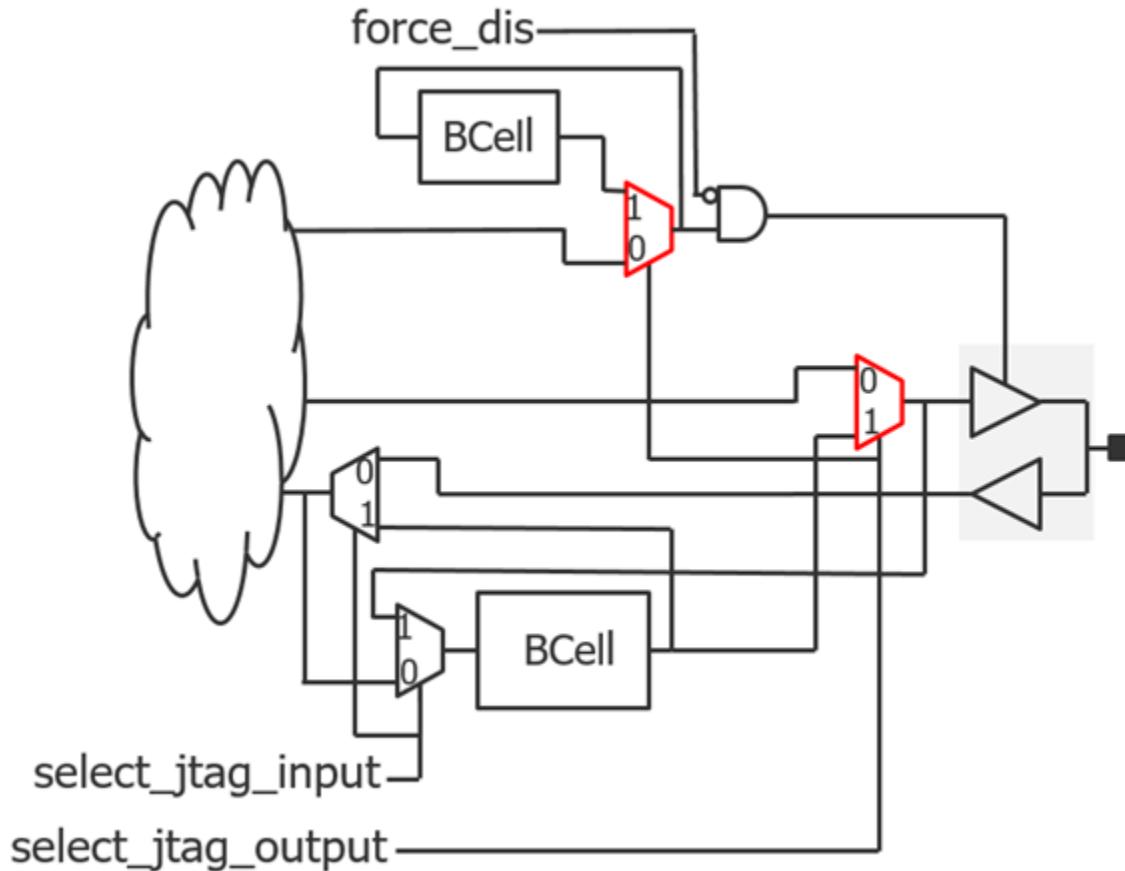


Figure 10-28 shows the boundary scan cell when it is used during ATPG. If you set the `max_segment_length_for_logictest` property found in the Boundary Scan wrapper to an integer or to unlimited, ATPG scan chains are created with the Boundary Scan chain. The blue path shows how the boundary scan cell is used to isolate the core values from the input ports by forcing `select_jtag_input` to 1 which is done by forcing the `int_ltest_en` pin on the boundary scan interface module. For more information about the boundary scan interface module and the logic inserted in it when the boundary scan chain is reused in logic test, see [Figure 10-35](#) on page 3391.

Note

If you are using the DFT signal flow, simply add the `int_ltest_en` DFT signal using the `add_dft_signals` command when you reach the top level and it will be connected to the boundary scan interface module. You can then activate the input isolation using the “`set_static_dft_signal_values int_ltest_en 1`” command and switch when it is time to create your ATPG patterns.

The `select_jtag_output` signal is under ATPG control allowing both the input of the output of the JTAG multiplexers (shown in red) to be tested. If you do not force the `int_ltest_en` pin to 1, then the `select_jtag_input` will also be under ATPG control and the input path can also be tested.

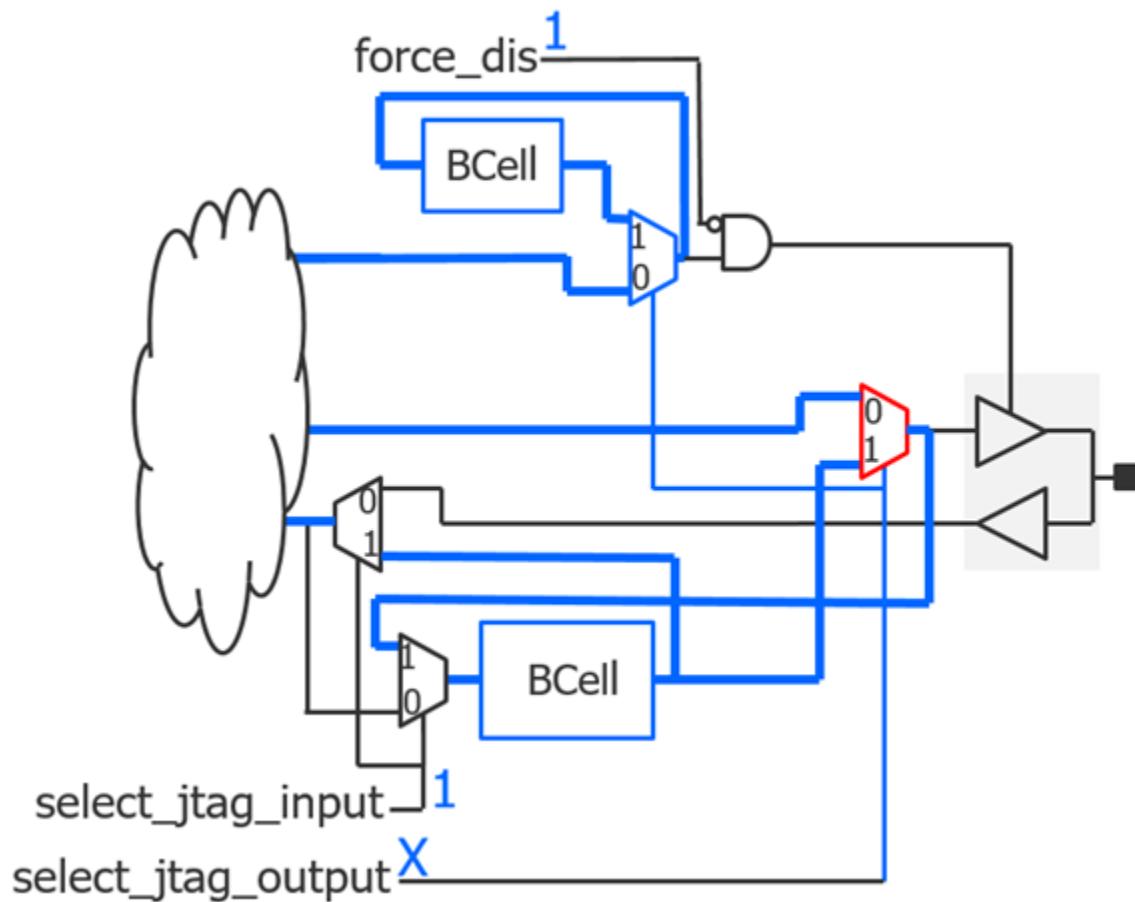
Figure 10-44. Boundary Scan Cell as Observation and Control in Logic Test Modes

Figure 10-29 shows the same boundary scan cell shown in Figure 10-27 but with auxiliary logic (colored green) added for both the input and output directions. Notice how the green multiplexer is not inserted along the functional path but on the test side of the JTAG multiplexer shown in red. The aux_in_en and aux_out_en signals are mutually exclusive, and the boundary scan cell is in functional mode when they are both low.

Figure 10-45. Adding Auxiliary Input and Output Logic in Bscan Cell

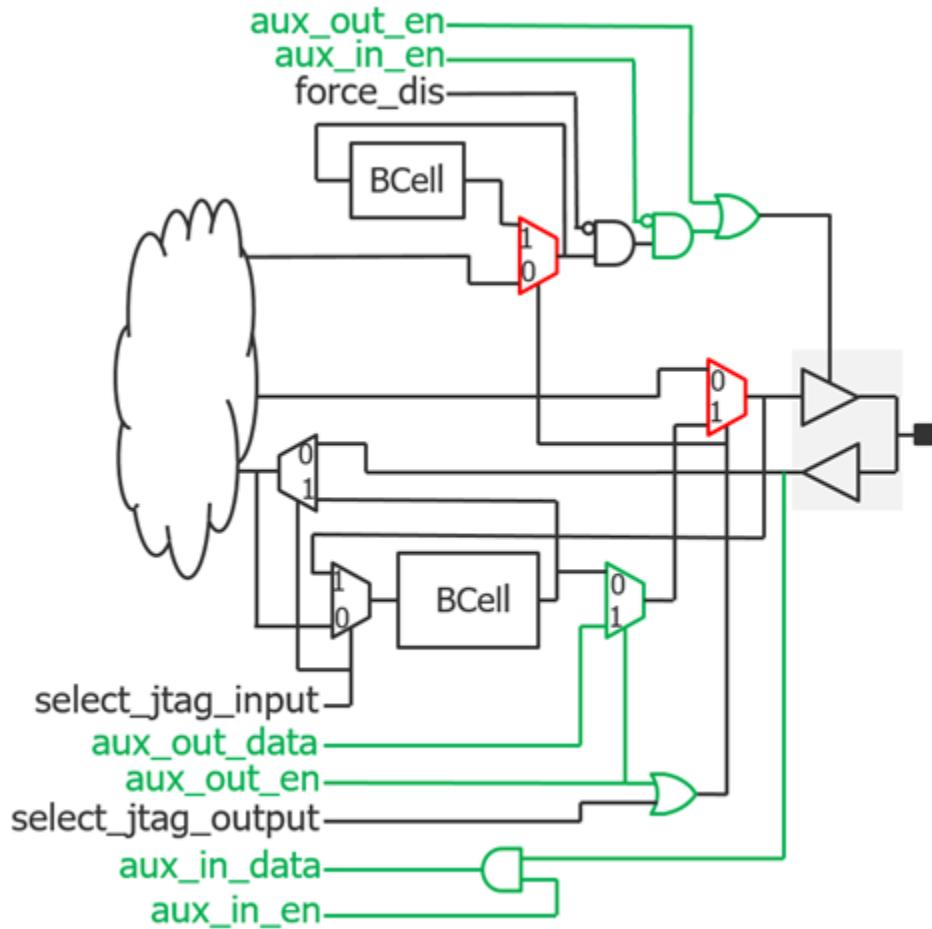


Figure 10-30 shows aux_out_en asserted to 1 to enable the auxiliary output data path. The two multiplexers are switched to select the aux_out_data value, and the enable of the output pad is forced on. When aux_out_en is high, the boundary scan cell is not able to observe the value coming from the core. This is why the [add_dft_modal_connections](#) command gates the aux_out_en signals with “scan_en” such that it is active only in shift mode. In capture mode, the aux_out_data is not needed, and instead the boundary scan cell is used to observe the functional core output. The capture mode is shown in Figure 10-31.

Figure 10-46. Path Used When AuxOut Logic Activated

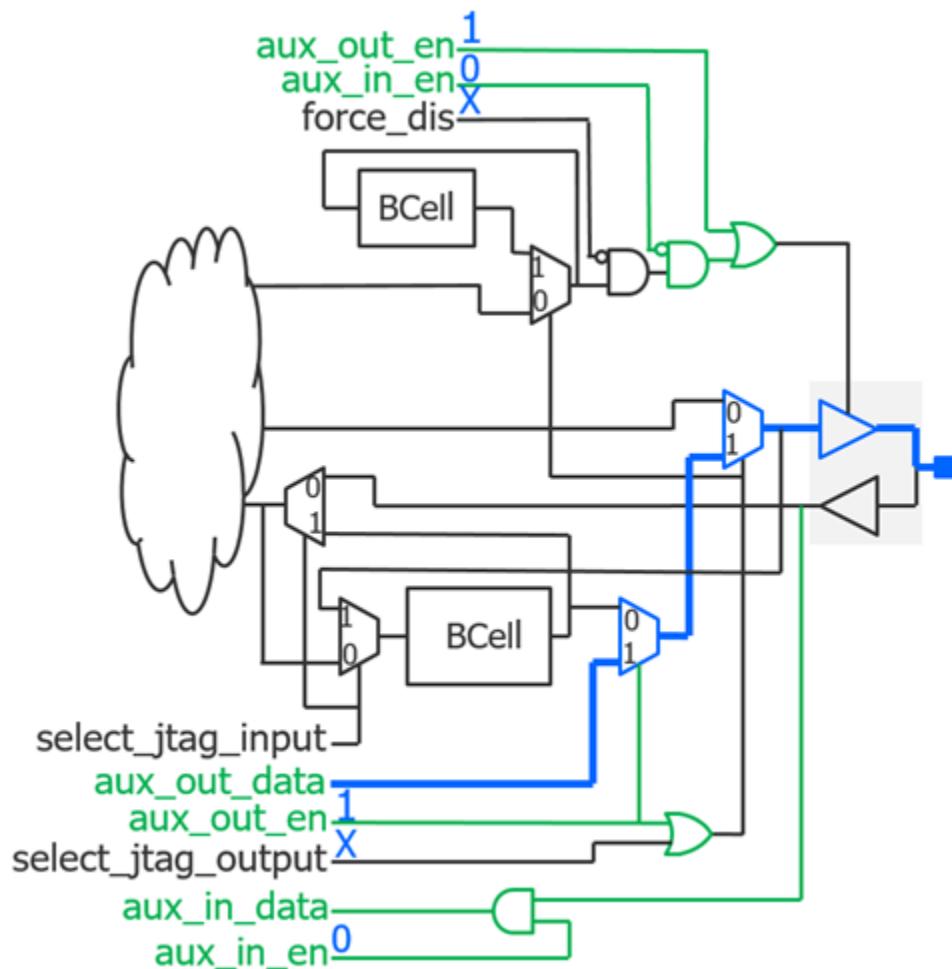


Figure 10-47. Forcing aux_out_en Low in Capture Mode to Observe Core Output

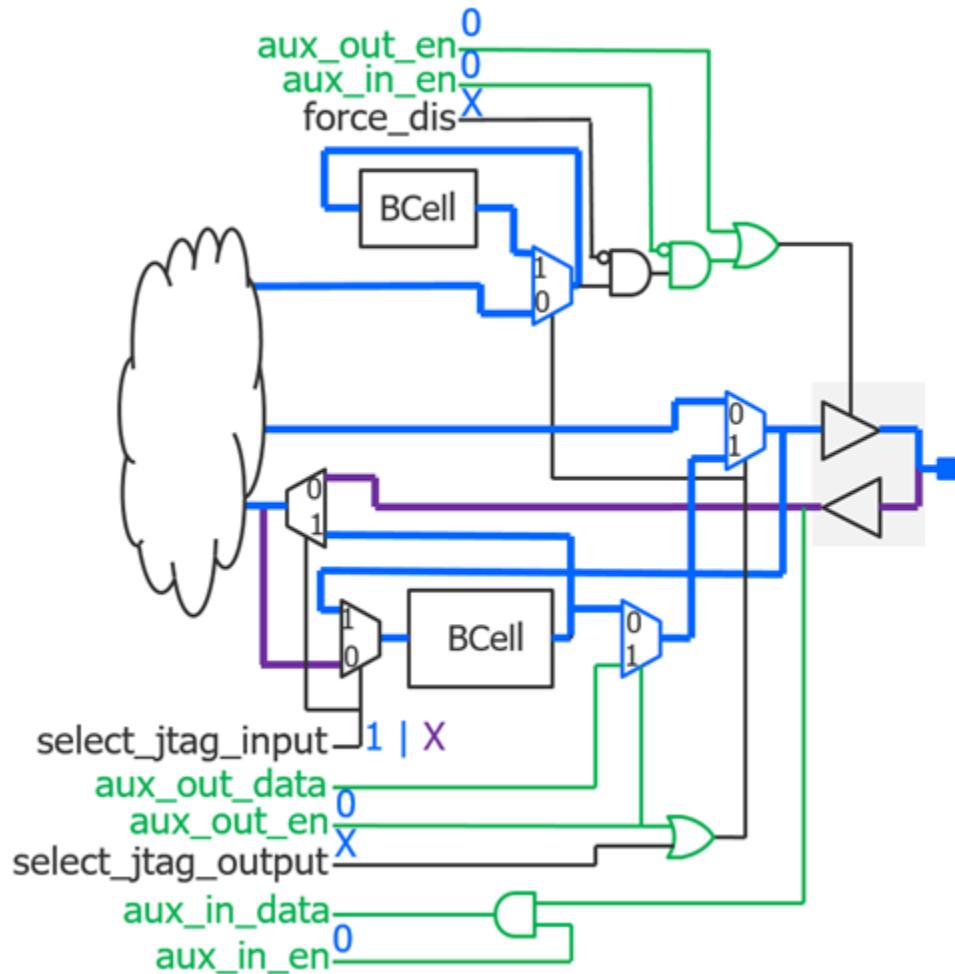


Figure 10-32 shows the IO pad used as an auxiliary input during ATPG. The boundary scan cell is used to control the signal going to the core while the input buffer provides the value to the auxiliary in data. This scheme is useful to allow reusing functional pins for the DFT signals `scan_en`, `test_clock`, and `edt_update` without losing ATPG fault coverage of those inputs.

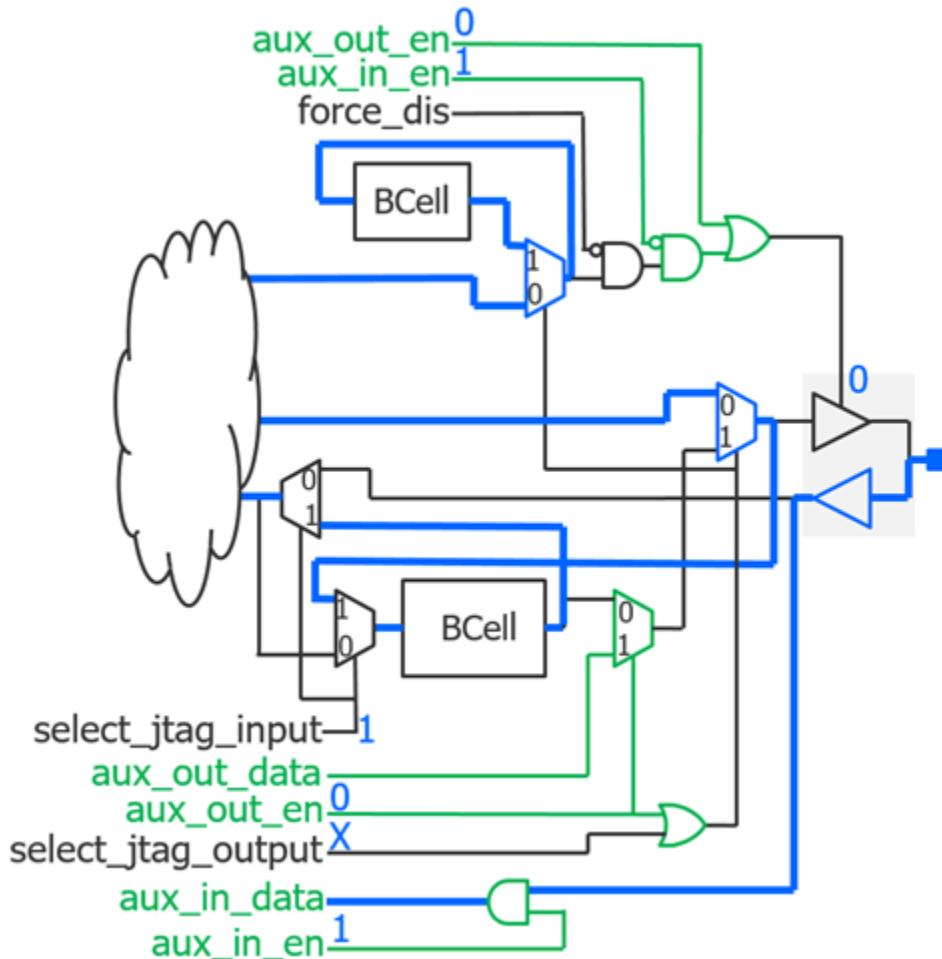
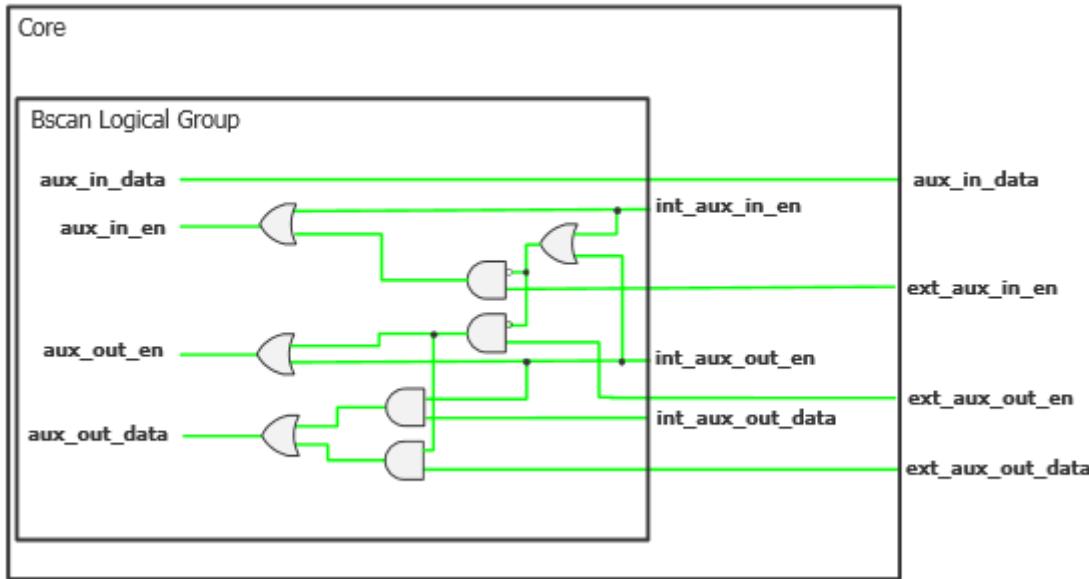
Figure 10-48. Using AuxIn and Controlling to Core Signal in ATPG

Figure 10-33 shows the extra logic that is added when the same port is used as both an internal and an external auxiliary data in the EmbeddedBoundaryScan/AuxiliaryInputOutputPorts wrapper. In the output direction, two data_out input pins are created on the logical grouping module. One of those inputs is used to make auxiliary output connections for elements found within the block and the other input is connected to an input of the block to make auxiliary output connections for elements found outside the block. In the input direction, the same data output port on the logical grouping module is used for providing auxiliary data inputs to elements located inside and outside the block. Both directions have an internal and external enable pin. The internal pins are used by the elements inside the blocks and the external pins are used by the elements outside the blocks. For example, if you call [add_dft_modal_connections](#) inside the block, it will make connections to the internal pins; and if you call [add_dft_modal_connections](#) outside the block, it will make connections to the external pins.

Figure 10-49. Extra Logic Added When a Port is Used Both Internally and Externally



Arguments

- auxiliary_input_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns to define a group of input or inout ports as auxiliary input ports. The same inout port can be both an input and an output auxiliary port. The auxiliary inputs are left tied on the logic group module to be used by test logic found inside the chip.

The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional, and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- auxiliary_output_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns that defines a group of output or inout ports as auxiliary output ports. The same inout port can be both an input and an output auxiliary port. The auxiliary output ports are left dangling on the logic group module to be used by test logic found inside the chip.

The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- internal_auxiliary_input_ports : *port_name_pattern*, ... ;

A property that specifies a list of port name patterns to define a group of input or inout ports as internal auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the

auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input and output ports of the block to be used by test circuit outside the block.

The *port_name_pattern* values can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `internal_auxiliary_output_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns that defines a group of output or inout ports as internal auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `external_auxiliary_input_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns to define a group of input or inout ports as external auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary output ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the [Interface](#)/`auxiliary_*` properties.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `external_auxiliary_output_ports : port_name_pattern, ... ;`

A property that specifies a list of port name patterns that defines a group of output or inout ports as external auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the [Interface](#)/`auxiliary_*` properties.

The *port_name_pattern* can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

Examples

The following example defines the GPIO00 to GPIO31 ports as both auxiliary input and output ports. The first 16 are usable by test logic that is both internal and external to the block. The last 16 are only used by test logic that is external to the block. The full flexibility is preserved such that each test mode will be able to use N of them as auxiliary output ports and the rest as auxiliary inputs where N can be different for each test mode.

```
DftSpecification(my_chip,rtl) {
    EmbeddedBoundaryScan {
        AuxiliaryInputOutputPorts {
            internal_auxiliary_input_ports : GPIO%2d[0:15] ;
            internal_auxiliary_output_ports : GPIO%2d[0:15] ;
            external_auxiliary_input_ports : GPIO%2d[0:31] ;
            external_auxiliary_output_ports : GPIO%2d[0:31] ;
        }
    }
}
```

EnableGroups

Specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        EnableGroups {
            EnableGroup(name) {
                capture_core_signal : on | off ;
                internal_enable_signal : pin_or_net_name ;
                port_list : port_name_pattern, ... ;
                insert_before_port : port_name ;
                insert_after_port : port_name ;
            }
        }
    }
}
```

Description

A wrapper that specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list. The properties `insert_before_port` and `insert_after_port` are used to specify the location of the enable cells associated with the group. The property `capture_core_signal` is used to prevent the cell from capturing the core signal during INTEST or during logictest when the boundary scan is used as isolation.

Arguments

- `capture_core_signal : on | off ;`

A property that specifies if the enable boundary-scan cells associated with the group are allowed to capture the core signal during INTEST or during the capture pulse when you reuse it for logictest isolation. See the `max_segment_length_for_logictest` property in the [BoundaryScan](#) wrapper for information about the logictest usage. When specified to off, the enable boundary-scan cell captures itself instead of capturing the core signal. This may be useful if you are using LogicBIST and the functional enable is X during the logictest mode.

- `internal_enable_signal : pin_or_net_name ;`

A property that references an internal signal inside the chip that directly controls the enable pin of a group of output pad cells. Specifying this enable signal is equivalent to listing all the ports controlled by that enable signal in the `port_list` property. The specification of the `internal_enable_pin` and `port_list` properties are mutually exclusive. If the number of output pad cells controlled by the enable signal is larger than the specified `outputs_per_enable_cell` property, the fanout of the enable cell will be split into smaller groups to meet the specified `outputs_per_enable_cell` value. One enable boundary-scan cell will be added per group.

- `port_list : port_name_pattern, ... ;`

A property that associates a group of output or inout ports with a given enable group. The `port_name_pattern` values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a

counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells specified in the list is larger than the specified outputs_per_enable_cell value, the list will be split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell will be added per group.

You do not need to specify an enable group for all output and inout ports. For the tri-state inout and output ports not explicitly associated to an enable group, one or more additional enable groups will be inferred. The inferring is done to match the common functional enable signal sourcing a group of pads. A given group can further be split to satisfy the specified outputs_per_enable_cell value described in the following sections:

- BoundaryScan — [outputs_per_enable_cell](#) property
- EmbeddedBoundaryScan — [outputs_per_enable_cell](#) property
- insert_before_port : *port_name* ;
A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.
- insert_after_port : *port_name* ;
A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

Examples

The following example defines three enable groups. The first one is associated with the ports in the fanout of the functional enable signal “core_i/en”. The second one is associated with all ports matching the pattern “OUT*”. The third group is associated with all other remaining ports as it does not specify either the internal_enable_signal nor the port_list property.

```
DftSpecification(my_chip,rtl) {
    BoundaryScan {
        EnableGroups {
            EnableGroup(g1) {
                internal_enable_signal : core1/en ;
            }
            EnableGroup(g2) {
                port_list : OUT* ;
            }
            EnableGroup(g3) {
            }
        }
    }
}
```

InternalBScanCells

Specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBScanCells {
            InternalBScanCell(name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                safe_value : 0 | 1 | X ;
                type : control | observation | both ;
                connection : pin_or_net_name ;
                multiplexing : on | off | auto ;
            }
        }
    }
}
```

Description

A wrapper that specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Arguments

- *name*

A string that uniquely identifies the internal boundary-scan cell in the BSDL file. The string starts with a letter and may be followed by any numbers of letters, numbers, or underscores. The string must be unique across all InternalBScanCell wrappers. The name of internal boundary-scan cells are mapped to the cell position inside the boundary scan chain using an attribute called TESSENT_INTERNAL_CELL_LABELS as shown in this example. The two InternalBScanCell wrappers have the names “my_name1” and “my_name2”.

```
-- BSDL local extensions
attribute TESSENT_INTERNAL_CELL_LABELS: BSDL_EXTENSION;
attribute TESSENT_INTERNAL_CELL_LABELS of TOP: entity is
    -- Label Cell
    "((my_name1 , 36 ) , " &
    " (my_name2 , 30 ))";
```

- *insert_before_port : port_name ;*

A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the *insert_before_port* property is mutually exclusive with the specification of the *insert_after_port* property. If neither the *insert_before_port* nor *insert_after_port* properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near *scan_out* in the order they were specified. The location

of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- `insert_after_port : port_name ;`

A property that specifies the position of the internal boundary-scan cell. This property can only be used when the `pin_order_file` property inside the [BoundaryScan](#) wrapper is specified. The specification of the `insert_after_port` property is mutually exclusive with the specification of the `insert_before_port` property. If neither the property `insert_before_port` nor `insert_after_port` properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near `scan_out` in the order they were specified. The location of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- `safe_value : 0 | 1 | X ;`

A property that specifies a safe value for the internal boundary-scan cell. This value is reflected into the BSDL file. If a non-X safe value is specified, the [BoundaryScan](#) signoff and manufacturing pattern set will never load a different value into that register during any of its [RunTest](#) specifications.

- `type : control | observation | both ;`

A property that specifies if the internal boundary-scan cell is to be used to control and/or observe an internal node. [Table 10-6](#) shows the hardware inserted into the design based on the value of the type and multiplexing property.

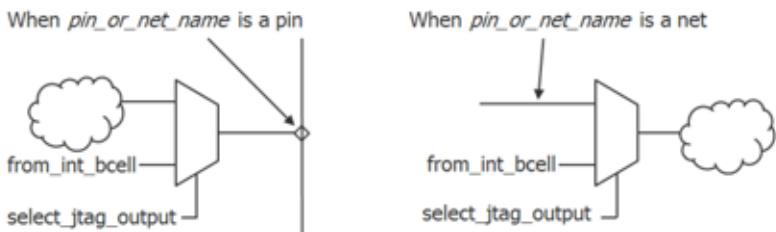
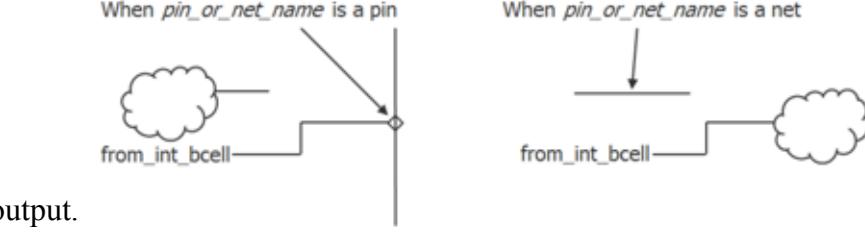
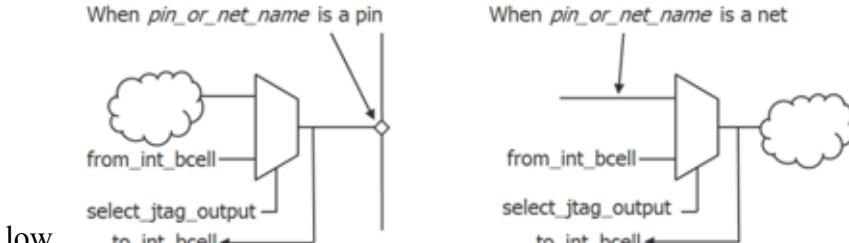
- `connection : pin_or_net_name ;`

A property that specifies the name of a pin or a net to control and/or observe. The `parent_instance` of `pin_or_net_name` will be auto-uniquified if not unique.

- `multiplexing : on | off | auto ;`

A property that is only relevant when the property `type` is set to “control” or “both”. A multiplexer is inserted as shown in [Table 10-6](#). When the value is “auto”, the multiplexer is inserted only when the `has_functional_source` attribute of the specified `pin_or_net_name` is set to “true”. See [Pin](#) and [Net](#) for more information about this attribute.

Table 10-10. Connections to and from Internal Boundary Scan Cells

type	description
control	<p>If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “true”, the driver is intercepted with a multiplexer located in the <i>parent_instance</i> of the <i>pin_or_net_name</i>. A net is intercepted at its source while a pin is intercepted at its destination. The internal boundary-scan cell captures its own output.</p>  <p>If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.</p> 
observation	The specified <i>pin_or_net_name</i> is connected to the input of the internal boundary-scan cell.
both	<p>If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “true”, the driver is intercepted with a multiplexer located in the <i>parent_instance</i> of the <i>pin_or_net_name</i>. A net is intercepted at its source while a pin is intercepted at its destination. The observation net is taken after the multiplexer. The functional source of the <i>pin_or_net_name</i> is observable when select_jtag_output is low.</p>  <p>If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for <i>pin_or_net_name</i> is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.</p>

Examples

The following example creates an internal boundary-scan cell that is used to control the u1/pll_reset signal using the boundary scan chain. The specified safe value of 0 ensures this cell will never be loaded with a 1 during any of the [RunTest](#) in the [PatternsSpecification](#) or with any board test patterns created from the BSDL file.

```
DftSpecification(module_name,id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBScanCells {
            InternalBScanCell(mycell) {
                insert_before_port : PortA ;
                safe_value         : 0 ;
                type              : control ;
                connection        : u1/pll_reset ;
                multiplexing      : auto ;
            }
        }
    }
}
```

InternalBscanSegments

Specifies the position of an existing internal boundary scan segment in the boundary scan chain.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan | EmbeddedBoundaryScan {
        InternalBscanSegments {
            InternalBScanSegment(instance_name) {
                insert_before_port : port_name ;
                insert_after_port : port_name ;
                logical_group_name : logical_group_name ;
            }
        }
    }
}
```

Description

A wrapper that is used to specify the position of an existing internal boundary scan segment in the boundary scan chain. When the BScanSegment contains cells associated with a port of the design, the position of the BScanSegment and its *logical_group_name* is taken from the position and the logical group of the first port associated with the BScanSegment in the *pin_order_file*. In the case of pure internal boundary scan segments, their position and logical grouping name cannot be extracted from associated ports because they are not associated with ports. The logical group name is not used to create logical grouping modules for existing boundary scan segments because they are already grouped into modules. Instead, they are used to refer to the segments that use the *bypass_logical_group* inside the [BondingConfigurations](#) wrapper.

This wrapper is automatically created when the [create_dft_specification](#) command is invoked if a BScanSegment description is found for a module. You simply need to delete this wrapper prior to invoking [process_dft_specification](#) if you do not want it to be part of the boundary scan chain. Refer to the “[set_design_sources -format bscan_lib](#)” command description for information about automatically finding the BScanSegment files without having to explicitly read them in using the [read_core_descriptions](#) command. You specify the module matching options using the [set_module_matching_options](#) command.

Arguments

- *insert_before_port* : *port_name* ;

A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the *insert_before_port* is mutually exclusive with the specification of the *insert_after_port* property. When neither the *insert_before_port* nor *insert_after_port* properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the *pin_order_file* property is used inside the [BoundaryScan](#) wrapper.

- `insert_after_port : port_name ;`

A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the `insert_before_port` is mutually exclusive with the specification of the `insert_after_port` property. When neither the `insert_before_port` nor `insert_after_port` properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the `pin_order_file` property is used inside the `BoundaryScan` wrapper.

- `logical_group_name : logical_group_name ;`

A property used to associate a `logical_group_name` to the existing internal boundary scan segment. This name is not used to create a logical grouping module as the existing boundary scan segment is already grouped into a module. It is used instead to refer to the segment using the `bypass_logical_group` property inside the [BondingConfigurations](#) wrapper.

Examples

The following example defines a design instance as an internal boundary scan segment. It is given a logical group name so that it can be bypassed in some bounding options. A `BscanSegment` definition must exist for the module associated with instance u1/u2.

```
DftSpecification(module_name,id) {
    BoundaryScan {
        InternalBScanSegments {
            InternalBScanSegment(u1/u2) {
                logical_group_name      : seg1 ;
            }
        }
    }
}
```

BondingConfigurations

Specifies different boundary scan chain configurations based on bonding variations of a single die into multiple packages.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        BondingConfigurations {
            BondingConfiguration(name) {
                enable_signal      : pin_or_net_name ;
                part_number_code   : binary ; // default: 16'h0
                version_code       : binary ; // default: 4'h0
                unused_ports       : port_name_pattern, ... ;
                bypassed_logical_groups : logical_group_name, ... ;
            }
        }
    }
}
```

Description

A wrapper that specifies different boundary scan chain configurations based on bonding variations of a single die into multiple packages. You need to have a one-hot set of decoded signals in which each signal is high for one bonding configuration and low for all the others. In a given bonding configuration, you can choose not to connect some port on the die to package pins. In this case, the boundary-scan cells are described as internal boundary-scan cells in the BSDL file associated with that bonding configuration unless you choose to group those cells in a logical grouping module and specify the module in the list of logical group modules to bypass. You typically need to bypass a segment of the boundary-scan cell when the segment is completely powered down in the given package.

You can also specify a different part_number_code and version_code for each of the bonding configurations. If you are using a third-party TAP controller, the controller must be capturing the part_number code and the version code from ports at the boundary of the TAP controller, and those ports must be described as DataInPorts, as described in the “[Requirements on a TAP to be usable for BoundaryScan](#)” section, and must be part of the CaptureSource of the device_id register.

Arguments

- *name*
A string that uniquely identifies each bonding configuration. This string is used in the name of the BSDL file associated with each bonding configuration.
- *enable_signal* : *pin_or_net_name* ;
A mandatory property that points to a signal inside the design that is high when the given bonding configuration is used and zero for all other bonding configurations.

If you specify an enable _signal, the tool issues the following warning:

```
// Warning: /DftSpecification(car,gate)/BoundaryScan/  
BondingConfigurations/BondingConfiguration(default)/enable_signal  
// You don't need to specify a enable signal. This entry will be  
// ignored.
```

An enable_signal is only needed if you define a bypassed_logical_groups, a part_number_code, or a version_code. A bonding configuration that has at least one of those needs an enable_signal. Conversely, a bonding configuration having none of those does not need one.

- part_number_code : *binary* ;

An optional property that defines the part_number_code to associate with the given bonding configuration. The generation of the part number code per bonding configuration is done inside the boundary scan interface module as shown in [Figure 10-34](#). If you are using a third-party TAP to host the boundary scan chain, it must have a part_number_code DataInPort and that port must be part of the CaptureSource of the device_id register as described in the “[Requirements on a TAP to be usable for BoundaryScan](#)” section.

If this property is not specified, its value is taken from the specified part_number_code property in the Tap wrapper referenced by the ijttag_host_node property of the [BoundaryScan](#) wrapper or from the tied value on the part_number_code DataInPort when using a TAP that already exists in the design.

- version_code : *binary* ;

An optional property that defines which version id code to associate with the given bonding configuration. The generation of the version id code per bonding configuration is done inside the boundary scan interface module as shown in [Figure 10-34](#). If you are using a third-party TAP to host the boundary scan chain, it must have a version_code DataInPort and that port must be part of the CaptureSource of the device_id register as described in the “[Requirements on a TAP to be usable for BoundaryScan](#)” section.

If this property is not specified, its value is taken from the specified version_code property in the Tap wrapper referenced by the ijttag_host_node property of the [BoundaryScan](#) wrapper or from the tied value on the version_code DataInPort when using a TAP that already exist in the design.

- unused_ports : *port_name_pattern*, ... ;

A property that specifies a list of ports that are not contacted in the given bonding configuration but whose boundary-scan cells remain part of the boundary scan chain. Those cells are described as internal boundary-scan cells in the BSDL. If the boundary-scan cells are powered down in the given bonding configuration, you must instead group the boundary-scan cells into one or many logical groups and specify that they are to be bypassed using the bypassed_logical_groups property.

The port_name_pattern can include an asterisk “*” as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum

number of digit the number must have. For example, ABC%2d[0:2] will match ABC00, ABC01, and ABC02.

- `bypassed_logical_groups : logical_group_name, ... ;`

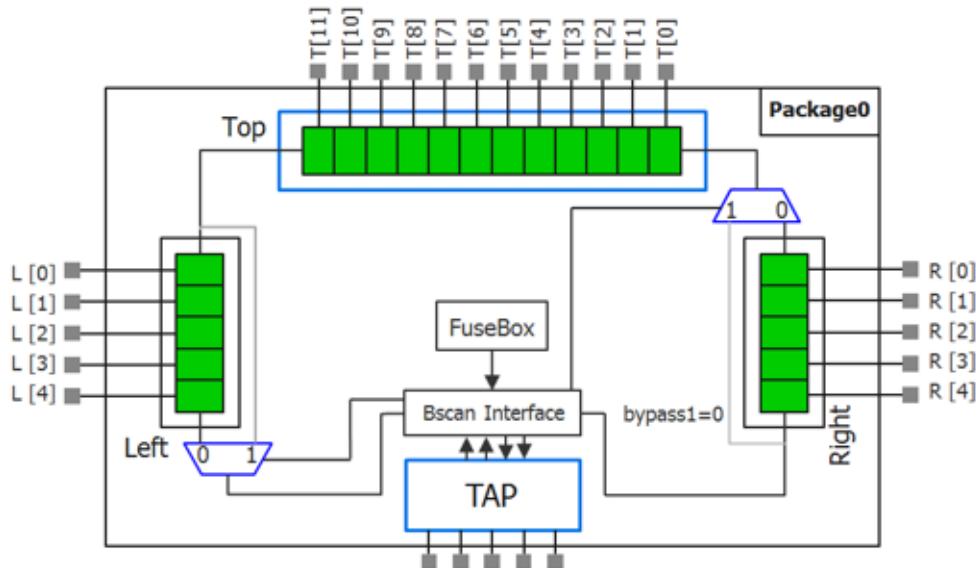
A property used to list one or more logical grouping module that are to be bypassed in the current bonding configurations. You use this option if you want to optimize the length of the boundary scan chain for each bonding configuration, or if the segment is completely powered down in the current bonding configuration. You can choose to declare the boundary-scan cells associated with the non-contacted ports as internal boundary-scan cells using the `unused_ports` property, but this is only possible if the boundary scan chain segment remains alive and powered in the current bonding configuration.

Examples

The following example shows three different bonding configurations.

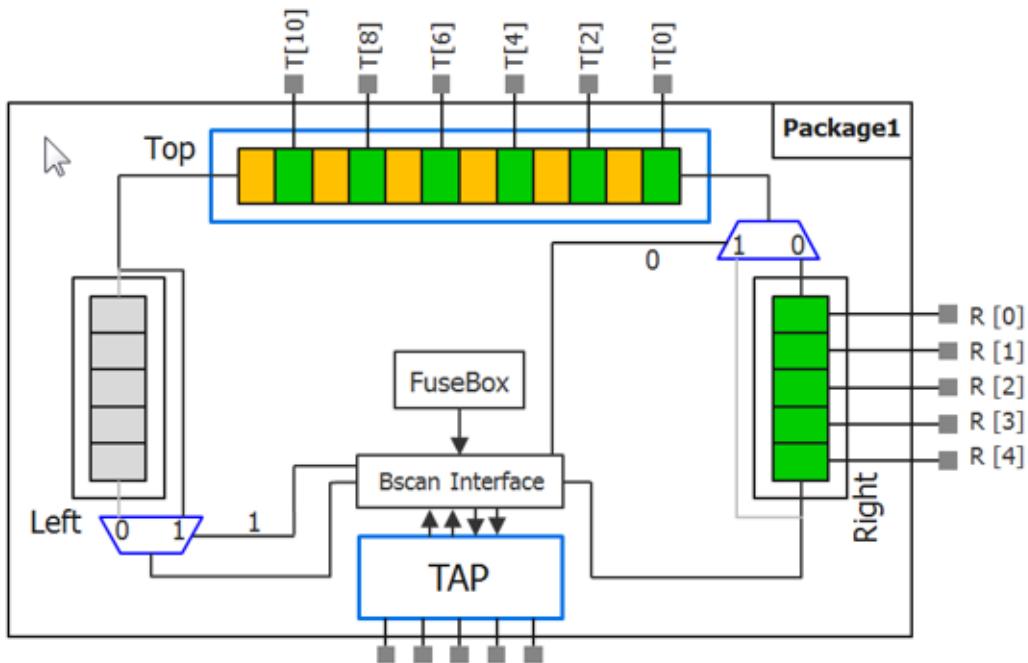
```
DftSpecification(my_chip,rtl) {
    BoundaryScan {
        ijtag_host_node : HostScanInterface(existing_tap) ;
        BondingConfigurations {
            BondingConfiguration (package0) {
                enable_signal           : core/bounding_decode/pkg0 ;
            }
            BondingConfiguration (package1) {
                enable_signal           : core/bounding_decode/pkg1 ;
                unused_ports            : L[*],T1,T3,T5,T7,T9,T11 ;
                part_number_code        : 16'h0001 ;
                bypassed_logical_groups : Left ;
            }
            BondingConfiguration (package2) {
                enable_signal           : core/bounding_decode/pkg2 ;
                unused_ports            : T6,T7,T8,T9,T10,T11 ;
                part_number_code        : 16'h0002 ;
                bypassed_logical_groups : Right ;
            }
        }
    }
}
```

The bonding configuration “package0” has no unused pins and no bypassed logical groups. The boundary scan register contains all boundary-scan cells in the design as shown in [Figure 10-50](#). No boundary-scan cells associated with unused ports are converted to internal boundary-scan cells. The BSDL file may contain internal boundary-scan cells if some were specified with the [InternalBScanCells](#) and [InternalBscanSegments](#) wrappers. Those cells are internal in all configurations unless they are bypassed. The part number code and the version code are not specified for this bonding configuration. Because this example uses a TAP already present in the design, the default values are taken from the tied values present on the `part_number_code` and `version_code` DataInPorts on the TAP where it is instantiated in the design.

Figure 10-50. package0 Bonding Configuration Example


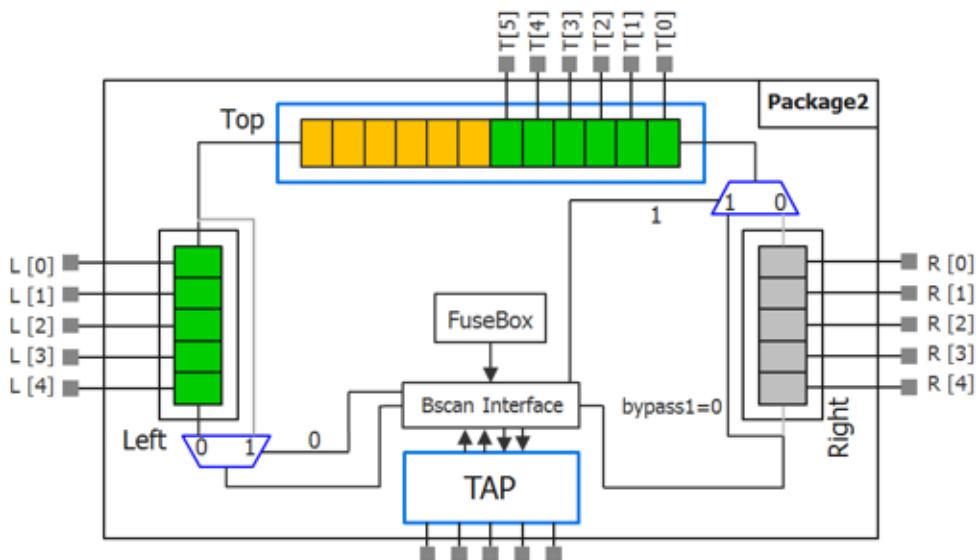
The bonding configuration “*package1*”, shown in [Figure 10-51](#), has several unused pins and a bypassed logical group called “Left”. The bypassed cells are shown in gray and the cells converted to internal cells are shown in orange. A different device_id code is specified for the configuration. Notice that the ports matching L[*] were specified with the unused_ports property. Those ports are already associated with a logical group that is declared as bypassed. This is a redundant specification. The cells associated with the L[*] ports will not be declared as internal boundary-scan cells in the BSDL file because they are bypassed. The cell associated with the ports T1,T3,T5,T7,T9,T11 are declared as internal boundary-scan cells as they are not bypassed.

Figure 10-51. package1 Bounding Configuration Example



The bonding configuration “package2”, shown in [Figure 10-52](#), has several unused pins and one bypassed logical group called “Right”. In this example, a set of fuses is used to program the bonding configuration. The one-hot encoded enable signals could have also been decoded from ports that get tied high and low in each respective package. When you want to simulate the BoundaryScan patterns using the various bonding configurations, you will need to make sure the right bonding configuration is enabled. See the description of the bonding_configuration property inside the BoundaryScan wrapper of the [PatternsSpecification](#) wrapper.

Figure 10-52. package2 Bonding Configuration Example



UserInstructions

Specifies a user instruction and an optional user register that is to be added in the BSDL file.

Usage

```
DftSpecification(module_name, id) {
    BoundaryScan {
        UserInstructions {
            UserInstruction(instr) {           // Repeatable
                tap_host_scan_interface_id : Tap(id)/HostIjtag(id) ;
                instruction_codes       : binary, ... ;
                bsd1_visibility          : public | private ;
                user_register             : user_reg[int] ;
            }
        }
    }
}
```

Description

An optional wrapper that is used to specify a user instruction and an optional user register that is to be added in the BSDL file. The UserInstruction wrapper *instr* name will be listed in the INSTRUCTION_OPCODE section of the BSDL file. If the *instr* is specified as being private, it will also be listed in the INSTRUCTION_PRIVATE section.

If a user register associated with *instr* is specified, it will be listed in the REGISTER_ACCESS section of the BSDL file. UserInstruction wrappers that specify user_register properties must have unique *user_reg* names or use the same *user_reg* name and *int* size.

The opcode(s) for the user instruction can be defined directly using the instruction_codes property, or they can be referenced from those defined in the [Tap/HostIjtag](#) wrapper for the current DftSpecification/[IjtagNetwork](#) using the tap_host_scan_interface_id property. Only one of these methods can be used for each UserInstruction wrapper.

Arguments

- UserInstruction(*instr*)

A repeatable wrapper used to specify the user instruction name. The UserInstruction wrapper *instr* name will be listed in the INSTRUCTION_OPCODE section of the BSDL file and must be a valid BSDL name.

- UserInstruction(*instr*)/tap_host_scan_interface_id : Tap(*id*)/HostIjtag(*id*) ;

A property that specifies the user instruction opcode(s) are to be referenced from those defined in the [Tap/HostIjtag](#) wrapper for the current DftSpecification/[IjtagNetwork](#). The Tap/HostIjtag wrapper must exist and the Tap(*id*) must be the TAP that hosts the boundary scan hardware. This property is mutually exclusive with the instruction_codes property.

- UserInstruction(*instr*)/instruction_codes : *binary*, ... ;

A property that specifies the binary opcode(s) associated with the *instr* user instruction. This property is mutually exclusive with the tap_host_scan_interface_id property.

- UserInstruction(*instr*)/bsdl_visibility : public | private ;

A property that specifies whether the instruction name *instr* is to only be used by the manufacturer, or to be available publicly. The default setting is public. If set to private, the *instr* will be listed in the INSTRUCTION_PRIVATE section of the BSDL file.

- UserInstruction(*instr*)/user_register : *user_reg[int]* ;

An optional property that specifies a user register that is associated with the user instruction *instr*. The parameter *user_reg* specifies the name of the user register and must be a valid BSDL name.

The parameter *int* specifies the size of the user register, and must be specified if the size cannot be determined from the tap_host_scan_interface_id or if the specified register is not one of BOUNDARY, BYPASS or DEVICE_ID. When tap_host_scan_interface_id is specified together with user_register, the parameter *int* will be checked against the selected part of the IJTAG network. If the selected part has a fixed length, it needs to match *int*. If the selected part controls a configurable scan path, or if the length cannot be determined, the user needs to specify the length as *int* in the user_register property. The parameter *int* is optional if the length can be determined from the selected IJTAG network.

The specified *user_reg* will be listed in the REGISTER_ACCESS section of the BSDL file. UserInstruction wrappers that specify user_register properties must have unique *user_reg* names or use the same *user_reg* name and *int* size.

If user_register is not specified, then bsdl_visibility must be set to private.

LogicBist

A wrapper that specifies the information used by the process_dft_specification command to build the Hybrid TK/LBIST IP and insert the IP into the design.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        ijtag_host_interface : name ; // default: tied_scan_interface
        Controller(id) {
        }
        NcpIndexDecoder {
        }
    }
}
```

Description

A wrapper that specifies the creation and the optional insertion of Hybrid TK/LBIST controllers in a design.

Arguments

- `ijtag_host_interface : name ;`

This property is used to specify the host scan interface that the Hybrid TK/LBIST controllers will connect to. You can specify the host name as specified by the HostScanInterface parameter within the [IjtagNetwork](#) wrapper with two exceptions: “None” is unsupported. Instead, use the `tied_scan_interface` parameter, which means that all IJTAG signals are tied to 0 and you have to properly reconnect those pins. You can use a post insertion script or do it manually after `process_dft_specification`.

Controller

This wrapper is used to add and specify properties for the Hybrid TK/LBIST controller generated with process_dft_specification.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            parent_instance           : name ;
            leaf_instance_name       : name ;
            burn_in                  : on | off ;
            pre_post_shift_dead_cycles : int ; // default: 8
            ControllerChain {
            }
            SingleChainForDiagnosis {
            }
            ShiftCycles {
            }
            CaptureCycles {
            }
            PatternCount {
            }
            WarmupPatternCount {
            }
            NcpOptions {
            }
            Interface {
            }
            Connections {
            }
        }
    }
}
```

Description

Use this wrapper to describe the shared Hybrid TK/LBIST controller instance. When using this wrapper, you must specify the [EDT](#) wrapper.

There is no way to specify multiple LBIST controllers and mapping of Hybrid TK/LBIST blocks to the LogicBIST controller. All EDT blocks specified in the same insertion pass will be hybrid IP connected with this controller.

Arguments

- `parent_instance : name ;`

This property is used to specify the parent instance. This is the name of the instance inside which you instantiate your hybrid IP. By default, the tool chooses the lowest common ancestor of all related EDT blocks.

- `leaf_instance_name : name ;`

This property is used to specify the leaf instance name.

- `burn_in : on | off ;`

This optional property specifies whether to enable burn-in test mode for the LogicBIST IP. The default is off. Refer to “[Burn-In Test Mode](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for details.

- `pre_post_shift_dead_cycles : int ;`

This optional property specifies how many dead cycles, at the shift clock period, will be applied before and after the capture clocks are pulsed. The specified integer can be between 2 and 8, with the default being 8. Reducing the number of pause cycles will reduce the test time but it may also make it harder to achieve timing closure of the test logic. To ensure that the shift and capture signals are staggered an integer value of 1 is not supported. Refer to “[Programmable Shift and Capture Pause Cycles](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for details.

ControllerChain

A wrapper that configures the Hybrid TK/LBIST Controller Chain Mode (CCM).

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            ControllerChain {
                present : on | off ;
                clock   : enum ; // legal    : tck (edt_clock)
            }
        }
    }
}
```

Description

A wrapper that configures the generation of ATPG patterns that target the Hybrid TK/LBIST logic. See “[Controller Chain Mode](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

Arguments

- **present : on | off ;**
This property specifies whether to enable or disable the CCM. When enabled, the tool creates an RTL scan chain so that it can calculate test coverage for the LogicBIST controller and Hybrid TK/LBIST blocks. The default is off.
- **clock : enum ;**
This property specifies whether the tool uses the tck or edt_clock clock during CCM. The default is edt_clock.

SingleChainForDiagnosis

A wrapper used to configure the hybrid TK/LBIST single chain mode logic.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            SingleChainForDiagnosis {
                present : on | off ;
                parent_instance : name ;
                leaf_instance_name : name ;
                skip_edt_blocks : on | off ;
            }
        }
    }
}
```

Description

You use this wrapper to configure the hybrid TK/LBIST single chain mode logic. See “[Single Chain Mode Logic](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

Arguments

- present : on | off ;

This property determines the “auto” value for the EDT/Controller/[BypassChains](#)/single_bypass_chain property. When the single_bypass_chain property is set to “auto,” it takes the value of the present property.

When the present property is “on” (the default), the EDT/Controller/BypassChains/single_bypass_chain property must be set to either “on” or “auto”. Tesson Shell generates an error if the single_bypass_chain property is off while the present property is on.

Note



Unlike many other options, if this property is set to “off”, then the parent_instance and leaf_instance_name properties are still meaningful. This is because the module controlling LBIST single chain for diagnosis is also responsible for any bypass logic of hybrid EDT blocks. Consequently, it will get instantiated even if the diagnostic logic for LBIST is disabled. Those properties will still determine the name of the module.

- parent_instance : *name* ;

This property is used to specify the parent instance under which the single chain for diagnosis logic will be instantiated. When unspecified, this property inherits its value from the ..parent_instance property.

- leaf_instance_name : *name* ;

This property is used to specify the leaf instance name.

- `skip_edt_clocks : on | off ;`

This property specifies whether to insert a SIB in each EDT block that allows optionally skipping all scan chains in a given EDT block during single chain unload. The default is on. For details, refer to “[Single Chain Mode Diagnosis](#)” in the *Hybrid TK/LBIST Flow User’s Manual*.

ShiftCycles

A wrapper the configures the shift cycles for the hybrid TK/LBIST controller.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            ShiftCycles {
                counter_resolution : enum ; // legal : byte bit
                max : int ;
                hardware_default : int ; // default: max
            }
        }
    }
}
```

Description

A wrapper the configures the shift cycles for the hybrid TK/LBIST controller.

Arguments

- counter_resolution : *enum* ;

This parameter specifies whether the shift counter allows shift lengths that operate in multiples of 8 or 1. The default, byte, causes the tool to round up the length of the longest scan chain to the next multiple of 8. When you specify “bit,” the tool applies the number of shifts equal to the length of the longest scan chain.

- max : *int* ;

This property specifies the maximum number of shift cycles for the controller. This is equivalent to the [set_lbist_controller_options](#) command’s “-max_shift_cycle” switch pair.

- hardware_default : *int* ;

This property specifies the default maximum number of shift cycles for the controller. The default is max.

CaptureCycles

A wrapper that configures the capture cycles for the hybrid TK/LBIST controller.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            CaptureCycles {
                max           : int ;
                hardware_default : int ; // default: max
            }
        }
    }
}
```

Description

A wrapper that configures the capture cycles for the hybrid TK/LBIST controller.

Arguments

- `max : int ;`

This property specifies the maximum number of capture cycles for the controller. This is equivalent to the [set_lbist_controller_options](#) command's “-max_capture_cycles” switch pair.

- `hardware_default : int ;`

This property specifies the default maximum number of capture cycles for the controller. The default is max.

PatternCount

A wrapper that configures hybrid TK/LBIST pattern counter.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            PatternCount {
                max           : int ;
                hardware_default : int ; // default: max
            }
        }
    }
}
```

Description

A wrapper that configures hybrid TK/LBIST pattern counter register.

Arguments

- `max : int ;`

This property specifies the maximum number of patterns for the controller. This is equivalent to the `set_lbist_controller_options` command's “-max_pattern_count” switch pair.

- `hardware_default : int ;`

This property specifies the default maximum number of patterns for the controller. The default is max.

WarmupPatternCount

This wrapper configures the hybrid TK/LBIST warm-up pattern count.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            WarmupPatternCount {
                max : int ; // default: 511
                hardware_default : int ; // default: 0
            }
        }
    }
}
```

Description

This wrapper configures the hybrid TK/LBIST warm-up pattern count. See “[Warm-Up Patterns](#)” in the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

Arguments

- `max : int ;`

This property specifies the maximum number of warm-up patterns for the controller. This is equivalent to the `set_lbist_controller_options` command’s “`-max_warmup_pattern_count`” switch pair. The default is 511.

- `hardware_default : int ;`

This property specifies the default maximum number of warm-up patterns for the controller. The default is 0.

NcpOptions

A wrapper that specifies NCP-related hardware options for the LogicBIST controller.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            NcpOptions {           // following two properties are mutually exclusive
                count : int ;     // default: 1
                percentage_of_patterns_per_ncp : integer, ... ;
            }
        }
    }
}
```

Description

This wrapper differs from the NcpIndexDecoder wrapper in that NcpOptions specifies NCP-related hardware parameters for the LogicBIST controller whereas you use the NcpIndexDecoder wrapper to configure the index decoder hardware, such as clock waveforms and OCCs.

Specify the NcpOptions wrapper when you are inserting LogicBist and the NCP Index Decoder in different passes. You must specify the count property or the percentage_of_patterns_per_ncp property. The latter property applies the NCPs for the specified percentages of patterns. The count and percentage_of_patterns_per_ncp properties are mutually exclusive.

When you generate the NcpIndexDecoder in the same pass as the LogicBIST controller, Tesson Shell automatically populates the NcpOptions/count parameter from the number of NcpIndexDecoder/Ncp wrappers. In this case, specifying the NcpOptions wrapper results in an error.

Arguments

- **count : int ;**
An integer that specifies to generate a LogicBIST controller that can support up to the specified integer of NCPs. The default is 1. The integer you specify must equal the number of NCPs as specified by the NcpIndexDecoder wrapper.
- **percentage_of_patterns_per_ncp : integer, ... ;**
A repeatable integer that represents a percentage value. This value specifies, as a percentage, the number of times an NCP will be applied every 256 patterns. The first NCP applies to the first integer specified in the percentage_of_patterns_per_ncp list, the second NCP to the second integer, and so on. The number of patterns each NCP is active in (within a 256 pattern range) is stored in the LogicBIST controller's hardware.

Interface

An optional wrapper the specifies the port names on the interface of the LogicBIST controller module.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller(id) {
            Interface {
                IjtagScanInterface {
                    tck : port_name; // default: ijtag_tck
                    reset : port_name; // default: ijtag_reset
                    select : port_name; // default: ijtag_sel
                    capture_en : port_name; // default: ijtag_ce
                    shift_en : port_name; // default: ijtag_se
                    update_en : port_name; // default: ijtag_ue
                    scan_in : port_name; // default: ijtag_si
                    scan_out : port_name; // default: ijtag_so
                }
                ControllerChain {
                    enable : port_name; // default: ccm_en
                    scan_in : port_name; // default: ccm_scan_in
                    scan_out : port_name; // default: ccm_scan_out
                    scan_en : port_name; // default: ccm_scan_en
                }
                test_clock : port_name; // default: test_clock
                edt_update : port_name; // default: edt_update
                shift_clock_src : port_name; // default: shift_clock_src
                clock_out : port_name; // default: edt_lbist_clock
                reset_out : port_name; // default: lbist_reset
                enable_out : port_name; // default: lbist_en
                done : port_name; // default: -
                prpg_en : port_name; // default: prpg_en
                misr_en : port_name; // default: misr_accumulate_en
                ijtag_select_out : port_name; // default: edt_sib_en
                scan_en : port_name; // default: scan_en[_in|_out]
            }
        }
    }
}
```

Description

The Interface wrapper is an optional wrapper defined within the LogicBIST controller wrapper that specifies the port names on the interface of the LogicBIST module. You can use this wrapper to rename the listed ports. Each property has a default value as described in the configuration data syntax.

Arguments

- IjtagScanInterface/tck : *port_name* ;

An optional property that specifies the name of the test clock IJTAG control signal. The default is ijtag_tck.

- IjtagScanInterface/reset : *port_name* ;
An optional property that specifies the name of the reset IJTAG control signal. The default is ijtag_reset.
- IjtagScanInterface/select : *port_name* ;
An optional property that specifies the name of the select IJTAG control signal. The default is ijtag_sel.
- IjtagScanInterface/capture_en : *port_name* ;
An optional property that specifies the name of the enable capture IJTAG control signal. The default is ijtag_ce.
- IjtagScanInterface/shift_en : *port_name* ;
An optional property that specifies the name of the enable shift IJTAG control signal. The default is ijtag_se.
- IjtagScanInterface/update_en : *port_name* ;
An optional property that specifies the name of the enable update IJTAG control signal. The default is ijtag_ue.
- IjtagScanInterface/scan_in : *port_name* ;
An optional property that specifies the name of the scan in IJTAG control signal. The default is ijtag_si.
- IjtagScanInterface/scan_out : *port_name* ;
An optional property that specifies the name of the scan out IJTAG control signal. The default is ijtag_so.
- ControllerChain/enable : *port_name* ;
An optional property that specifies the name of the controller chain enable port. This port is only used when DftSpecification/LogicBist/Controller/SingleChainForDiagnosis/present is set to on. The default is ccm_en.
- ControllerChain/scan_in : *port_name* ;
An optional property that specifies the name of the controller chain scan in port. This port is only used when DftSpecification/LogicBist/Controller/SingleChainForDiagnosis/present is set to on. The default is ccm_scan_in.
- ControllerChain/scan_out : *port_name* ;
An optional property that specifies the name of the controller chain scan out port. This port is only used when DftSpecification/LogicBist/Controller/SingleChainForDiagnosis/present is set to on. The default is ccm_scan_out.
- ControllerChain/scan_en : *port_name* ;
An optional property that specifies the name of the controller chain scan enable port. This port is only used when DftSpecification/LogicBist/Controller/SingleChainForDiagnosis/present is set to on. The default is ccm_scan_en.

- **test_clock : *port_name* ;**
An optional property that specifies the name of an input clock port with which to operate the LogicBIST controller. The default is test_clock.
- **shift_clock_src : *port_name* ;**
An optional property that specifies the name of an input clock port with which to operate the LogicBIST controller. The default is shift_clock_src.
- **edt_update : *port_name* ;**
An optional property that specifies the name of an input clock port that connects to the edt_update signal. This port is only created when the clock gater—created by running add_dft_signal shift_capture_clock –create_from_other_signals—is present. The default is edt_update.
- **clock_out : *port_name* ;**
An optional property that specifies the name of an output port for the generated LogicBIST clock. The default is lbist_clock.
- **reset_out : *port_name* ;**
An optional property that specifies the name of an output port that indicates when the synchronous reset of LogicBIST controller occurs. The default is lbist_reset.
- **enable_out : *port_name* ;**
An optional property that specifies the name of an output port that indicates when LogicBIST is enabled. The default is lbist_en.
- **done : *port_name* ;**
An optional property that specifies the name of an output port that indicates when the LogicBIST controller has finished its test. This port is only created when a port name is provided.
- **prpg_en : *port_name* ;**
An optional property that specifies the name of an output port that indicates when the PRPG is active during the scan load/unload. The default is prpg_en.
- **mistr_en : *port_name* ;**
An optional property that specifies the name of an output port that indicates when the MISR is to accumulate the scan unload data. The default is mistr_accumulate_en.
- **ijtag_select_out : *port_name* ;**
An optional property that specifies the name of the IJTAG ToSelectPort that enables the EDT IJTAG network. The default is edt_sib_en.
- **scan_en : *port_name* ;**
An optional property that is used to create the input and output ports that are connected to the source of an intercepted scan enable signal. The tool appends the “_in” and “_out”

suffices to the specified port name to create the input and output ports. The defaults are scan_en_in and scan_en_out.

Examples

The following example renames LogicBIST ports back to their pre-2018.1 release versions for purposes of backwards annotation.

```
DftSpecification(<module_name>,<id>) {
    LogicBist {
        Controller(<id>) {
            Interface {
                IJtagScanInterface {
                    tck      : tck;
                    reset   : sib_reset;
                    select   : sib_en;
                    capture_en : sib_capture_en;
                    shift_en   : sib_shift_en;
                    update_en   : sib_update_en;
                    scan_in     : lbist_scan_in;
                    scan_out    : lbist_scan_out;
                }
                ControllerChain {
                    enable   : ccm_en
                    scan_in  : ccm_scan_in
                    scan_out : ccm_scan_out
                    scan_en  : ccm_scan_en
                }
                test_clock      : edt_clock;
                clock_out       : lbist_clock;
                reset_out       : lbist_reset;
                enable_out      : lbist_en;
                prpg_en         : prpg_en;
                misr_en         : misr_en;
                ijttag_select_out : edt_sib_en;
                scan_en         : scan_en;
                shift_clock_src : shift_clock_src;
            }
        }
    }
}
```

Connections

This wrapper is used to add and specify properties for the hybrid TK/LBIST controller connections.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        Controller {
            Connections {
                edt_clock : port_pin_name ;
                    // default: OptionalDftSignal(edt_clock)
                edt_update : port_pin_name ;
                    // default: OptionalDftSignal(edt_update)
                tck : port_pin_name ;
                shift_clock_src : port_pin_name ;
                scan_en_in : port_pin_name ;
                    // default: OptionalDftSignal(scan_en)
                controller_chain_enable : port_pin_name ;
                    // default: control_chain_enable
                controller_chain_scan_in : port_pin_name ;
                    // default: control_chain_scan_in
                controller_chain_scan_out : port_pin_name ;
                    // default: control_chain_scan_out
            }
        }
    }
}
```

Description

This wrapper is used to add and specify properties for the hybrid TK/LBIST controller connections.

Arguments

- `edt_clock : port_pin_name ;`

Note

 When using dft signal for `edt_clock`, it must be sourced from an existing design pin rather than created from other signals. That is, you must specify the `edt_clock` signals with the “`add_dft_signal edt_clock -source_node`” command rather than the “`add_dft_signal edt_clock -create_from_other_signals`” command.

An optional property used to specify a source node to connect the `edt_clock` port to. The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the `set_design_level` command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion. When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(edt_clock)` or `DftSignal(edt_clock)` as the value, and it will be

interpreted to request a connection to the DFT signal `edt_clock` as it was added using the [add_dft_signals](#) command. When using `OptionalDftSignal(edt_clock)`, you will not get an error if the DFT signal `edt_clock` was not added; a connection to the port called “`edt_clock`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

- `edt_update : port_pin_name ;`

An optional property used to specify a source node to connect the `edt_update` port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(edt_update)` as the value, and it will be interpreted to request a connection to the DFT signal `edt_update` as it was added using the [add_dft_signals](#) command. When using `OptionalDftSignal(edt_update)`, you will not get an error if the DFT signal `edt_update` was not added; a connection to the port called “`edt_update`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value defaults to `OptionalDftSignal(edt_update)`.

- `tck : port_pin_name ;`

An optional property used to specify TCK. When the design level is other than “chip”, then by default the value of this property is the `ijtag_tck` pin as specified by the `ijtag_host_interface` parameter. If the design level is “chip”, this parameter defaults to “`tck`”.

- `shift_clock_src : port_pin_name ;`

A mandatory property used to specify the shift clock source. A dash (-) value indicates that the shift clock source is not desired, in which case, only TCK and the `edt_clock` will be allowable shift clock sources.

- `scan_en_in : port_pin_name ;`

An optional property used to specify scan enable for mux-DFF type (input wrapper cells).

- `controller_chain_enable : port_pin_name ;`

An optional property used to specify the controller chain enable.

- `controller_chain_scan_in : port_pin_name ;`

An optional property used to specify the controller chain scan in.

- `controller_chain_scan_out : port_pin_name ;`

An optional property used to specify the controller chain scan out.

NcpIndexDecoder

Decodes the NCP index output of the hybrid TK/LBIST controller into clock sequences to be generated by the Tesson OCCs across all capture procedures.

Usage

```
DftSpecification (module_name, id) {
    LogicBist {
        NcpIndexDecoder {
            parent_instance : instance_name ;
            leaf_instance_name : name ;
            Interface {
                ncp_index : port_name ; // default: ncp_index
                clock_sequence : port_name ; // default: occ%d_clock_sequence
            }
            Connections {
                ncp_index : port_pin_constant_name, ... ;
            }
            Ncp(id) {
                cycle(index) : occ_spec, ... ; // repeatable
            }
        }
    }
}
```

Description

The LBIST NCP index decoder is a simple combinational logic block that decodes the NCP index output of the LBIST controller into clock sequences to be generated by the OCCs across all capture procedures.

Processing the DftSpecification of the LogicBist wrapper using the process_dft_specification command produces the following outputs, which are written into the TSDB directory:

- Verilog RTL for the index decoder.
- Test procedure files for NCP description. It is recommended the timeplate you use during LBIST fault simulation uses the pulse_clock in the timeplates. This way, you can avoid the need for editing the timeplate to add the clock names used in the tool generated NCPs.
- Dofile that describes the mapping between the LBIST NCP index and the NCP names. This file should be sourced during LBIST fault simulation.

See “[NCP Index Decoder](#)” in the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- parent_instance : *instance_name* ;

Instance path name of the parent in which the NCP index decoder is instantiated. The default is the top-level of the design.

- `leaf_instance_name : name ;`
Name of the leaf instance.
- `Interface/ncp_index : port_name ;`
Name of the decoder's NCP index input. The default is `ncp_index`.
- `Interface/clock_sequence : port_name ;`
The default is `occ%d_clock_sequence`.
- `Connections/ncp_index : port_pin_constant_name, ... ;`
Points to the LBIST controller's NCP output. The specified pins are connected to the NCP index decoder input. When not specified, the NCP index decoder input pins are tied to "0".
- `Ncp(id)`
Repeatable wrapper that describes a single NCP. A capture procedure could either describe activity in one clock while others are OFF, or across multiple clocks.
- `Ncp(id)/cycle(cycle_number) : occ_spec, ... ;`
Repeatable named property that specifies the clock sequence for a cycle within an NCP.
The `cycle_number` is an integer starting from 0, which corresponds to the first cycle of capture. The clock activity on the first cycle is parallel loaded into the LSB of the OCC shift register, the second cycle onto the next significant bit and so on. When a particular cycle number is omitted, it means there is no clock activity for that cycle. The largest cycle number that can be specified is one less than the width of the OCC control Shift Register. The output NCP contains cycles from 0 till the largest cycle number specified, including empty cycles.

The `occ_spec` is a repeatable list of Tesson OCC specification that can include:

- OCC instance path names.
- OCC module names.
- OCC core names used in TCD.
- OCC controller IDs generated in the same pass.
- Literal string "bscan_occ" to indicate a mini-occ inside boundary scan logic.
- Literal string "sti_occ" to indicate sib(sti) logic.

Ambiguity in specification (for example, a module and a top level instance with same name, or a top level instance and instance leaf name with same names) are resolved in the above order. Avoid potential ambiguity by using the full instance path names.

The tool issues an error if you specify only one `Ncp` wrapper, and it does not generate the NCP index output.

Examples

Example 1

In this example, assume the design has two top level OCC instances named *m8051_gate_tessent_occ_clk1_inst* and *m8051_gate_tessent_occ_clk2_inst* of the same OCC module *m8051_gate_tessent_occ*.

```
DftSpecification {
    LogicBist {
        NcpIndexDecoder {
            Connections {
                ncp_index: m8051_lbist_i/ncp ;
            }
            Ncp(stuck) {
                cycle(0): m8051_gate_tessent_occ ;
                //Specified using module name, refers to both the OCC instances.
            }
            Ncp(clk1_double_pulse) {
                cycle(0): m8051_gate_tessent_occ_clk1_inst ;
                cycle(1): m8051_gate_tessent_occ_clk1_inst ;
            }
            Ncp(clk2_double_pulse) {
                cycle(0): m8051_gate_tessent_occ_clk2_inst ;
                //Note - cycle(1) is omitted, so no clock activity
                cycle(2): m8051_gate_tessent_occ_clk2_inst ;
            }
        }
    }
}
```

Example 2

The following example inserts OCC and the NCP index decoder at the same time. As shown below, you must specify *occ_spec* with the ID as specified for the Occ/Controller that you are generating in the same pass.

```
DftSpecification(m8051, rtl)  {
    Occ {
        Controller(1) {                                     // Controller ID = 1
            clock_intercept_node: NX1g/Z;
        }
        Controller(2) {                                     //Controller ID = 2
            clock_intercept_node: NX2g/Z;
        }
    }
    LogicBist {
        NcpIndexDecoder {
            Ncp(slow) {
                cycle(0): Occ(1);                      // Occ/Controller generated above
                cycle(0): Occ(2);                      // Occ/Controller generated above
            }
            Ncp(NX1_fast) {
                cycle(0): Occ(1);
                cycle(1): Occ(1);
            }
            Ncp(NX2_fast) {
                cycle(0): Occ(2);
                cycle(1): Occ(2);
            }
        }
    }
}
```

LpctType3

As part of the process_dft_specification flow, the LpctType3 DftSpecification syntax defines the configuration of a low pin count test (LPCT) Type3 controller to be generated and inserted into the design.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
        }
        Connections {
        }
        Controller {
        }
    }
}
```

Description

You use an LPCT Type3 controller when there is no scan enable at the top level of the design and the design is not using a JTAG TAP controller for test. The LPCT Type3 controller generates the scan enable signal and all EDT specific control signals and you must use it in combination with an EDT wrapper. This controller requires an OCC instrument present in the design. OCC instruments may be specified in the configuration data or added as TCD instruments with the add_core_instances command. The LPCT Type3 controller will use and drive OCCs from a mix of OCC configuration data and TCD instrument instances. It also supports third-party OCCs that are added using the LpctType3/Connections/[ThirdPartyOCC](#) configuration data wrapper.

The inserted LPCT Type3 controller generates the scan enable and all the EDT specific control signals that drive the EDT and OCC controllers. This creates a restriction for interfaces used on the EDT and OCC modules; none of the instruments driven by the LPCT controller can have an IJTAG compatible interface. An error will be reported during validation if an EDT or OCC instrument that has an IJTAG compatible interface and is used in the PDS flow in combination with the LPCT.

The tool generates the RTL files for the LPCT Type3 controller inside the [Tessent Shell Data Base \(TSDB\)](#) when the LpctType3 wrapper is present in the configuration data and the process_dft_specification command is used. During the generation phase, the tool appends entries for the created LPCT files and modules to the synthesis dictionary inside the TSDB instrument container. This allows for the automated synthesis of the LPCT RTL files with the [run_synthesis](#) command.

The LPCT controller is typically used when the design has only 3 top level ports available for “logic test”: a free-running clock, an EDT channel input port, and an EDT channel output port. By default, the data_in connection of the LPCT controller instance is shared with the EDT channel input connection. If pipeline stages are used for EDT input channel connections, there will be no pipeline stages in the data path of the LPCT controller.

For more information, see [LPCT Controller Types](#) in the *TestKompress User's Manual*.

Arguments

None.

Examples

Example 1

This example shows a minimal set of configuration data that will create an LPCT Type3 Controller.

```
DftSpecification(CoreA,rtl1) {
    EDT {
        Controller(1) {
            ijtag_host_interface : none;
            longest_chain_range : 60, 100;
            scan_chain_count : 2;
            input_channel_count : 1;
            output_channel_count : 1;
        }
    }
    OCC {
        ijtag_host_interface : none;
        Controller(1) {
            clock_intercept_node : CLK;
        }
    }
    LpctType3 {
        Controller {
            max_capture_cycles : 5;
        }
    }
}
```

Example 2

In this example, an LPCT Type3 controller is specified and OCC core instances are added.

```

// command: add_core_instances -module CoreA_rtl1_tessent_occ
// Note: Reading TCD file './tsdbA/instruments/CoreA_rtl1_occ.instrument/
CoreA_rtl1_tessent_occ.tcd' from TSDB.
// Added core instance 'CoreA_rtl1_tessent_occ_1_inst'.
// Added core instance 'CoreA_rtl1_tessent_occ_2_inst'.
// Added core instance 'CoreA_rtl1_tessent_occ_3_inst'.
// command: read_config_data -from_string {
    DftSpecification(CoreA,rtl2) +{
        EDT {
            Controller(1) {
                scan_chain_count: 100;
                longest_chain_range: 100,110;
                input_channel_count: 2;
                output_channel_count: 2;
            }
            Controller(2) {
                scan_chain_count: 100;
                longest_chain_range: 100,110;
                input_channel_count: 2;
                output_channel_count: 2;
            }
        }
        LpctType3 {
            Controller {
                max_shift_cycles: auto;
                max_capture_cycles : 5;
                load_unload_cycles : 1, 2 ;
                max_scan_patterns : 100 ;
                max_chain_patterns : 10 ;
            }
        }
    }
}

```

Interface

A wrapper that specifies the port names for the LPCT Type 3 controller module.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
            clock : port_name ; // default: lpct_clock
            clock_mux_select : port_name ; // default: lpct_clock_mux_select
            capture_enable : port_name ; // default: lpct_capture_en
            shift_enable : port_name ; // default: lpct_shift_en
            shift_clock : port_name ; // default: lpct_shift_clock
            reset : port_name ; // default: lpct_reset
            scan_enable : port_name ; // default: lpct_scan_en
            data_in : port_name ; // default: lpct_data_in
            test_mode : port_name ; // default: lpct_test_mode
            reset_out : port_name ...; // default: lpct_reset_out
            reset_polarity : active_high | active_low ;
            test_end : port_name ; // default: lpct_test_end
            test_active : port_name ; // default: lpct_test_active
        }
        Connections {
        }
        Controller {
        }
    }
}
```

Description

You can use this wrapper to specify the names of the LPCT Type 3 controller ports. Each property has a default value, as described in the configuration data syntax. You can override the default name if you provide a value to be used for the port names of the LPCT controller module.

Arguments

- **clock : *port_name* ; // default: lpct_clock**
Specifies the name of the input clock for the LPCT controller. The default name is lpct_clock.
- **clock_mux_select : *port_name* ; // default: lpct_clock_mux_select**
Specifies the LPCT clock mux select port name. The default is lpct_clock_mux_select.
- **capture_enable : *port_name* ; // default: lpct_capture_enable**
Specifies the LPCT capture enable port name. The default is lpct_capture_enable.
- **shift_enable : *port_name* ; // default: lpct_shift_en**
Specifies the LPCT shift clock enable port name. The default is lpct_shift_en.

- `shift_clock : port_name ; // default: lpct_shift_clock`
Specifies the LPCT shift clock port name. The default is lpct_shift_clock.
- `reset : port_name ; // default: lpct_reset`
Specifies the LPCT reset port name. The default is lpct_reset.
- `scan_enable : port_name ; // default: lpct_scan_en`
Specifies the LPCT scan enable port name. The default is lpct_scan_en.
- `data_in : port_name ; // default: lpct_data_in`
Specifies the LPCT data in port name. The default is lpct_data_in.
- `test_mode : port_name ; // default: lpct_test_mode`
Specifies the LPCT test mode port name. The default is lpct_test_mode.
- `reset_out : port_name ; // default: lpct_reset_out`
Specifies the LPCT reset output port name. The default is lpct_reset_out.
- `reset_polarity : active_polarity ; // active_high active_low`
An optional switch and literal pair that specifies the activity state for the input reset signal of the generated LPCT controller. The default is active_high.
- `test_end : port_name ; // default: lpct_test_end`
Specifies the LPCT test_end port name. The default is lpct_test_end.
- `test_active : port_name ; // default: lpct_test_active`
Specifies the LPCT test_active port name. The default is lpct_test_active.

Connections

This wrapper specifies the LPCT Type3 controller connections.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
        }
        Connections {
            clock      : port_pin_name ; // default: lpct_clock
            reset      : port_pin_name ; // default: lpct_reset
            scan_enable : port_pin_name ;
            data_in     : port_pin_name ; // default: lpct_data_in
            test_mode   : port_pin_name ; // default: lpct_test_mode
            reset_out   : port_pin_name, ... ;
            test_end    : port_pin_name ;
            test_active : port_pin_name ;
            ThirdPartyOCC {
            }
        }
        Controller {
        }
    }
}
```

Description

The connections wrapper for the LPCT Type3 controller identifies its connections, which you specify with port or pin names. When the LPCT Type3 controller is inserted into the chip design top-level, all of the connections with a specified value must have an existing top-level port that is connected to a PAD cell. An error will be reported if this is not the case.

When the LPCT Type3 controller is inserted in a physical block or sub block design, the specified connections are created to the specified ports or pins. If the ports do not exist, they will be created.

Arguments

- `clock : port_pin_name ;`

Required argument that specifies an always-pulse clock for the LPCT. Default clock name is lpct_clock.

- `reset : port_pin_name;`

An optional property that specifies the input reset signal connection for the LPCT controller. The reset pin can be shared with the design's functional reset. This can be either a top-level port or an internal power-on-reset pin. The default reset name is lpct_reset.

You can manually specify the input and output reset connections for the LPCT controller. Use the `reset` property to specify the input reset connection to the controller and the `reset_out` property to specify the connection points that the LPCT controller will reset.

- **scan_enable : *port_pin_name*;**
An optional argument for LPCT controllers, that specifies the list of pins to which the LPCT output scan enable pin is connected.
- **data_in : *port_pin_name*;**
Optional argument that specifies the pin or port connected to the LPCT controller data_in pin. For the LPCT controller, a test sequence and configuration is scanned in through this port. This port is usually shared with the edt_channel input. The default data in name is lpct_data_in.
- **test_mode :*port_pin_name*;**
Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used. Default test mode name is lpct_test_mode.
- **reset_out : *port_pin_name*, ...;**
Specifies the output connection, or connections, of the LPCT reset signal sent to the design. The value of this property can be a bus or a list of connection points. The tool makes the specified connections from the output reset pin of the controller and connects the reset port or pin to the input reset pin of the controller.
- **test_end : *port_pin_name*;**
Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used.
- **test_active : *port_pin_name*;**
Optional argument that specifies an output signal from that LPCT controller that indicates the LPCT controller is active and controlling the scan test.

ThirdPartyOCC

A wrapper that specifies connections for a third-party OCC. If any of the properties is specified in this wrapper indicates a third-party OCC is used in the design.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
        }
        Connections {
            ThirdPartyOCC {
                clock_mux_select : port_pin_name ;// default: lpct_clock_mux_select
                capture_enable   : port_pin_name ; //default: lpct_capture_en
                shift_enable     : port_pin_name ; //default: lpct_shift_en
                shift_clock      : port_pin_name ; //default: lpct_shift_clock
            }
            Controller {
            }
        }
    }
}
```

Description

Third-party wrappers are specified only within the LpctType3/Connections wrapper. Use this wrapper to specify properties for a third-party OCC.

Arguments

- **clock_mux_select : *port_pin_name* ;// default: lpct_clock_mux_select**
An optional argument that specifies the connection to be made to the `clock_mux_select` output of the LPCT controller. This pin is a MUX select signal that selects between the shift clock generated by the LPCT controller and the capture clock. If `clock_mux_select` is high, shift clock is selected; if `clock_mux_select` is low, the capture clock is selected.
- **capture_enable : *port_pin_name* ;// default: lpct_capture_en**
A property that specifies the connection for the `capture_enable` output signal of the LPCT controller.
- **shift_enable : *port_pin_name* ;// default: lpct_shift_en**
A property that specifies the connection for the `shift_enable` output signal of the LPCT controller.
- **shift_clock : *port_pin_name* ;// default: lpct_shift_clock**
A property that specifies the connection for the `shift_clock` signal of the LPCT controller.

Controller

A wrapper used to specify all the LPCT Type3 specific settings.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
        }
        Controller {
            max_shift_cycles      : int | auto ;
            max_capture_cycles   : int ;
            load_unload_cycles   : count1, count2 ;
            max_scan_patterns    : int ;
            max_chain_patterns   : int ;
            shift_control_type   : enable | clock | none ;
            TestModeDetect {
            }
            parent_instance       : instance ;
            leaf_instance_name   : instance ;
        }
    }
}
```

Description

The controller wrapper specifies all the LPCT Type3 IP specific settings. There can only be one controller wrapper within the LpctTye3 wrapper, therefore the tool only creates a single LPCT controller during a single process_dft_specification pass.

Arguments

- `max_shift_cycles : integer;`

A property that specifies the maximum number of shift cycles for the LPCT controller. By default the value of this property is set to auto, which indicates that the maximum number of shift cycles is calculated automatically by the tool. The value of this property affects the size of the LPCT shift cycle counter register. If it is manually set, it needs to be large enough so the LPCT shift cycle counter can handle the longest shift length (from all EDT controllers). If is is too small, an error is reported during the validation phase.

- `max_capture_cycles : integer ;`

A property that specifies the maximum number of capture cycles for all test patterns. This is a required property.

- `load_unload_cycles : count1, count2;`

A property that specifies the number of cycles that precede and follow the shift cycles in the load_unload procedure. The value count1 is the number of cycles that precede the shift cycles in the load_unload procedure, while the value for count2 is the number of cycles that follow the shift cycles in the load_unload procedure.

- `max_scan_patterns : integer ;`
A property that specifies the maximum number of scan test patterns generated in the test application.
- `max_chain_patterns : integer;`
A property that specifies the maximum number of chain test patterns generated in the test application.
- `shift_control_type : {enable | clock | none};`
A property that specifies how the LPCT controller generates the shift clock control signal. By default the value of this property is set to enable. The enable value is the only value allowed when using Tessent OCCs. When you use third-party OCCs, the value of this property can be set to any value allowed by the syntax.
- `parent_instance : instance`
A property that specifies the name of the parent instance in which the LPCT is instantiated. By default, the LPCT controller is instantiated on the top level.
- `leaf_instance_name : instance;`
A property that specifies the name of the LPCT controller instance. By default the instance name follows this pattern:
 `${design_name}_ ${design_id}_ tessent_ ${instrument_type}`

TestModeDetect

A wrapper used to configure how the LPCT enters the test mode.

Usage

```
DftSpecification(module_name, id) {
    LpctType3 {
        Interface {
            }
            Connections {
            }
            Controller {
                TestModeDetect {
                    type           : <enum> ; // legal : (signal) sequence
                    input_sequence : <binary_seq>, <cycle_count> ;
                }
            }
        }
    }
}
```

Description

The LPCT controller can enter test mode in one of two ways: by detecting a specific input sequence within a particular number of cycles after test controller reset, or by a signal that indicates test mode. These two ways of entering test mode are mutually exclusive.

Use this wrapper to specify the controller enable type: signal or sequence. By default, the tool will use the test_mode connection from the Connections wrapper to cause the controller to enter test mode. If you select the sequence detection type, you must specify the input sequence that places the LPCT controller into test mode. The test controller enters test mode by detecting a specific sequence within a particular number of cycles or by a signal that indicates test mode.

Arguments

- type : signal | sequence ;

A property that specifies the LPCT controller enable type, sequence or signal. By default, the value is set to signal, indicating that the tool will use the test_mode connection from the Connections wrapper.

When using a test mode signal, test mode will start after the signal is asserted. This signal needs to stay asserted for only one cycle, after which it can be de-asserted at any time. The test session will not end when this signal is de-asserted; instead it will end only after end of test as determined by the pattern counters. In this case, it is your responsibility to ensure test mode will not be entered during functional operation. When this signal is a top level pin, the tool can generate the correct test_setup procedure. When this signal is an internal pin, you must modify the test setup procedure for pattern generation to ensure that the internal test mode signal will be asserted as desired.

To use sequence detection, you specify the input sequence and the number of cycles within which it should be detected during IP creation. Doing this ensures the chip will not enter test

mode during functional operation. To use sequence detection, must specify the input sequence property.

When the input test procedure already has an existing test_setup, the sequence detect period should also consider the additional cycles necessary for applying test_setup.

- **input_sequence** : *binary_seq* , *cycle_count* ;

A property that specifies the binary sequence that triggers the LPCT test mode if it is detected within the number of specified cycles after the LPCT controller has been reset.

MemoryBISR

Specifies the creation and the optional insertion of BISR registers.

Usage

```
DftSpecification(module_name,id) {
    MemoryBISR {
        bisr_segment_order_file : file_name ;
        memory_repair_loading_method : serial | from_read_buffer ;
        memory_library_name_list : mem_lib_name, ... ;
        BisrElement(memory_instance) {
        }
        Interface {
        }
        Controller {
        }
    }
}
```

Description

A wrapper that specifies the creation and the optional insertion of BISR registers. The presence of the controller wrapper adds a controller in the current design. A chip is allowed to have more than one Bisr controller as long as you have a set of fuses for each controller. Refer to “[Built-In Repair Analysis](#)” in the *Tessent MemoryBIST User’s Manual* for a detailed introduction to the topic, including what your memory library needs to include in order to perform Built-In Repair Analysis, and which [Interface](#) your fuse box needs to have to be usable by the controller.

Arguments

- `bisr_segment_order_file : file_name ;`

This property is used to reference a file that contains the power domain and ordering definition of the BISR segments using the syntax shown [Figure 10-53](#). This file is typically auto-generated when the [check_design_rules](#) command runs unless the [set_memory_instance_options -bisr_segment_order_file](#) command was used in which case the specified file is not generated but validated.

This property is typically needed unless the [process_dft_specification](#) command is run with the `-no_insertion` option; if this option is not used, the property is optional.

When the [check_design_rules](#) command runs, a `bisr_segment_order_file` is generated when all of these conditions are met:

- The [set_dft_specification_requirements -memory_test](#) option was specified as “on”.
- The [set_dft_specification_requirements -memory_bisr_chain](#) option was not set to off.
- The [set_memory_instance_options -bisr_segment_order_file](#) option was not specified.

- At least one memory instance exists with a Memory library description that has at least one Memory/.../SpareElement wrapper present *and the* [set_memory_instance_options -use_in_memory_bisr_dft_specification](#) option was not set to “off” for that instance, or, alternatively, at least one block with an ICL file that describes one or more BISR chains is present and the [ignore_for_dft_specification](#) pin attribute is not set to “on” for at least one of the BISR scan-in pins. A BISR scan-in pin as described in ICL has a [ScanInPort](#) that is part of a [ScanInterface](#) with a [tessent_bisr_chain_length Attribute](#) set to a value greater than 0.

The bisr_segment_order_file is created using the format shown in [Figure 10-53](#). If you load a DEF file using the [read_def](#) command, the elements in the OrderedElements wrapper will be ordered so as to minimize wire lengths between them. If you load a UPF or CPF file using the [read_upf](#) or [read_cpf](#) commands, a PowerDomainGroup wrapper will be created for each power domain name that has at least one element. You can also use the [Instance](#) and [Pin](#)-level attribute “bisr_power_domain_name” to define the power domains. When both the “power_domain_name” and the “bisr_power_domain_name” attributes are defined on the same instance, the latter wins. The advantage of the “bisr_power_domain_name” attribute is that it can be set differently on different BISR scan-in pins to associate them to different power domain names. You can also use the attribute to create extra BISR power domains that are not defined in the UPF/CPF files.

When the [check_design_rules](#) command runs and the following conditions are met:

- The [set_dft_specification_requirements -memory_test](#) option was specified as “on”
- The [set_dft_specification_requirements -memory_bisr_chain](#) option was not set to off
- The [set_memory_instance_options -bisr_segment_order_file](#) option was specified to refer to an existing file

The tool validates the referred bisr_segment_order_file to make sure that it lists all existing BISR chain elements and only existing BISR chain elements. An existing BISR chain element is a memory instance with a Memory library description that has at least one Memory/.../SpareElement wrapper (see “[Tessent Core Description](#)” on page 3755) present and the [set_memory_instance_options -use_in_memory_bisr_dft_specification](#) option was not set to “off” for that instance or a BISR scan-in pin on a block having an ICL file describing the BISR chains and the [Pin](#) attribute called “[ignore_for_dft_specification](#)” is not set to “on” on that pin. A BISR scan-in pin is described in ICL and has a [ScanInPort](#) that is part of a [ScanInterface](#) with the [Attribute](#) “[tessent_bisr_chain_length](#)” set to a value greater than 0.

When running the [process_dft_specification](#) command, the validation of the file pointed to by the bisr_segment_order_file property is repeated again to make sure that it lists all existing BISR chain elements and only existing BISR chain elements. If the property is not specified, an empty file is assumed and an error will be generated for all missing existing BISR chain elements.

The format of the BISR segment order file is shown in [Figure 10-53](#). When referring to an instance of a block that has more than one BISR scan segment, you must add the name of the scanin port to distinguish one chain from the other. The <block_instance_name>/<bisr_scanin_port_name> paths must point to pins of an instance with an associated ICL file where the Scanin Port is described as a [ScanInPort](#) that is part of a [ScanInterface](#) with the [Attribute](#) “tessent_bisr_chain_length” set to a value greater than 0. The [extract_icl](#) command automatically sets this attribute when generating an ICL module that contains BISR chains. The BisrModules wrapper is used when the [process_dft_specification](#) command is run with the -no_insertion option and it is desired to generate BISR registers for the memory specified by BisrModules/<modulename>. BisrModules/<modulename> must correspond to a memory core description that is loaded into memory using the [read_core_descriptions](#) command or the [set_design_sources](#) -format tcd_memory command. When running process_dft_specification with the -no_insertion option, the module names of repairable memories must be provided by the user because there is no design information that can be referenced to obtain this information.

Figure 10-53. BISR Segment Order File

```
BisrSegmentOrderSpecification {
    PowerDomainGroup(power_domain_name) { // Repeatable
        OrderedElements {
            memory_instance_name;
            block_instance_name[/bisr_scanin_port_name];
        }
    }
    BisrModules {
        modulename; // Repeatable
    }
}
```

- `memory_repair_loading_method : serial | from_read_buffer ;`

When this property is set to its default value of “serial”, the repair information is read from the fuse box, decompressed and then sent serially to the repair registers located near the memories. The advantages are that a relatively small number of fuses and very few wires are required which simplifies layout. The disadvantage is the longer length of time required to load the repair registers. It might be necessary to reduce the number of repair registers by enabling sharing of repair logic. See the [repair_sharing](#) property, and other related properties, in the [MemoryCluster](#) wrapper of the DftSpecification.

When this property is set to “from_read_buffer”, a multiplexer is inserted between the BISR module and the memory repair ports. This multiplexer allows the repair information to be sourced from either the fuse box read buffer or from the BISR registers. When using the read buffer as the repair source, the repair information is driven from the fuse box read buffer and broadcast to all memory repair ports in parallel. The advantages are that the repair information is available very quickly at the memory inputs and there is no limit on the number of incremental repair sessions. The disadvantages are that the memory repair information is stored in an uncompressed form and a lot of connections are created between the BISR controller and the memories. Therefore, a lot more fuses are required and routing

congestion might occur during layout. For these reasons, it might be necessary to reduce the number of repair registers by enabling sharing of repair logic. See the `repair_sharing` property, and other related properties, in the [MemoryCluster](#) wrapper of the `DftSpecification`.

The multiplexer select signal is connected to a test data register, which is controllable during patterns generation.

When generating the fuse box controller, a new signal call “SerialRepairEnable” is present on the fuse box controller if the design contains BISR chains that support fast BISR loading.

- `memory_library_name_list : mem_lib_name, ... ;`

A property that specifies a list of memory libraries associated with repairable memories in the design. This property is only used and needed if you are running the `process_dft_specification` command with the `-no_insertion` option and the design is not elaborated in memory. Otherwise, the `memory_library_name` is extracted from the design for the memory instances indicated inside the file specified in `bisr_segment_order_file`.

BisrElement

Overrides the settings of the BISR registers and specifies their location.

Usage

```
DftSpecification(module_name, id) {
    MemoryBisr {
        BisrElement(memory_instance) { // repeatable
            parent_instance : relative_instance ;
            parent_instance_anchor : current_design |
                parent_instance_of_memory | attribute_expression ;
        }
    }
}
```

Description

A repeatable wrapper that is used to override settings of the BISR registers for repairable memory instances and specify their insertion into a particular parent instance of the design. If memory instances are matched by multiple BisrElement wrappers due to wildcards, the last BisrElement property in the DftSpecification will override any prior wrappers.

Note

 When relocating the BISR modules inside non-unique design instances, Tessent Shell will attempt to preserve design uniqueness. However, if the relocation of the BISR modules causes non-unique instances to differ, Tessent Shell will unify the design instances in order to satisfy the BISR relocation requirements.

If the BisrElement wrapper is not specified, or if it is specified but has no properties specified within the wrapper, the BISR registers will be inserted adjacent to their respective memory instances.

Arguments

- *memory_instance*
A pattern expression matching repairable memory instances in the design. The wildcard character “*” may be used to select multiple memory instances.
- *parent_instance : relative_instance ;*
A property that specifies a design instance relative to the instance specified with *parent_instance_anchor*. If not specified, the default is a null string which will instantiate the BISR register in the same design hierarchy as specified by the *parent_instance_anchor* property.
- *parent_instance_anchor : current_design | parent_instance_of_memory | attribute_expression ;*
This property is only valid if *parent_instance* is specified. The *parent_instance_anchor* property specifies the starting point, or anchor, for the relative path specified with the

relative_instance value of the *parent_instance* property. The default value of *current_design* specifies the top level of the current design. Specifying *parent_instance_of_memory* will set the anchor one level up in the design hierarchy from the memory instance(s) matched by *memory_instance*. The *attribute_expression*, enclosed in double quotes, is used to find a design instance on which the attribute expression is true. The expression is evaluated beginning with the parent instance of the memory. If the expression is true, this instance is set as the *parent_instance_anchor*. If the expression is false, the next level parent instance is then evaluated. This continues until the expression is true or the top level of the design is reached.

Examples

Example 1

The following example shows how the BISR register instances for the repairable memory instances memA and memB, originally matched by a wildcard pattern, are placed into the desired top-level design locations by subsequent BisrElement wrappers.

```
BisrElement(*) {
    parent_instance_anchor : current_design ;
    parent_instance : block_instC;
}
BisrElement(core_inst*/memA) {
    parent_instance_anchor : current_design ;
    parent_instance : block_instA;
}
BisrElement(core_inst*/memB) {
    parent_instance : block_instB;    // parent_instance_anchor defaults to
}                                // current_design if not specified.
```

If memA and memB instances are located throughout the design hierarchy, the instances located within these blocks will match the subsequent BisrElement *memory_instance* expression:

```
core_instA/memA, core_instB/memA, core_instC/memB, core_instA/memB
```

The final location for the BISR register instances will then be as follows:

```
block_instA          // all matched memA BISR registers
block_instB          // all matched memB BISR registers
block_instC          // all globally matched memory BISR registers
```

Example 2

The following example shows how the BISR register instances for the repairable memory instance memA, which was originally matched by a wildcard pattern, are restored to the same design instance as the memory through a final match for that memory instance:

```
BisrElement(*) {  
    parent_instance_anchor : current_design ;  
    parent_instance : block_instA;  
}  
BisrElement(*/memA) {  
    parent_instance_anchor : parent_instance_of_memory ;  
    parent_instance : "";  
}
```

If memA is located in the design hierarchy at:

```
top/block_instB/memA
```

The final location for the BISR register instances will then be at:

```
top/block_instB           // memA BISR registers adjacent to memA  
block_instA               // BISR registers for all other memories
```

Example 3

This example outlines another use of the parent_instance_of_memory option for the parent_instance_anchor property with a more elaborate parent_instance specification. Assume the design contains a memory instance memA, located in the design hierarchy at:

```
block_instA/c_inst1/d_inst/memA
```

The following BisrElement wrapper is defined to instantiate the BISR registers for memA:

```
BisrElement(*) {  
    parent_instance_anchor : parent_instance_of_memory ;  
    parent_instance : ../../memA_bisr_container_inst;  
}
```

The parent instance of memA is: block_instA/c_inst1/d_inst, which will be the anchor. The parent_instance property may contain “..” sequences to define relative paths from the parent instance anchor. The single “..” sequence in this example moves up one level in hierarchy from the anchor, then appends the instance memA_bisr_container_inst. The final location for memA BISR instances will then be:

```
block_instA/c_inst1/memA_bisr_container_inst
```

Example 4

The following example locates each BISR register at the first parent level that has the “power_domain_island” attribute equal to “pdgA_2”. The evaluation of the expression starts

from the parent instance of each matched memory and works up the design hierarchy. point to attribute filtering equation syntax section for more.

```
BisrElement(*) {  
    parent_instance_anchor : "power_domain_island==pdgA_2" ;  
}
```

If no match is found, the BISR registers will be placed at the top level of the design.

Note that any of the listed [Instance](#) built-in attributes, or any user attributes that are [registered](#) against a design instance, can be used within the *attribute_expression*. Additionally, any of the [Attribute Filtering Equation Syntax](#) relations can be implemented to create more complex expressions.

Interface

Specifies the names of the ports created when connecting the BISR chains to ports of a module of type `sub_block` or `physical_block` as specified using the `set_design_level` command.

Usage

```
DftSpecification(module_name, id) {
    MemoryBisr {
        Interface {
            scan_in           : port_naming ; //Def: %s_bisr_si
            scan_out          : port_naming ; //Def: %s_bisr_so
            capture_shift_clock : port_naming ; //Def: %s_bisr_clk
            shift_en          : port_naming ; //Def: %s_bisr_shift_en
            reset              : port_naming ; //Def: %s_bisr_reset
            parallel_in       : port_naming ; //Def: %s_bisr_parallel_in
            serial_repair_enable: port_naming ;
                                         //Def: %s_bisr_serial_repair_enable
            memory_disable     : port_naming ; //Def: %s_bisr_mem_disable
            memory_chain_select: port_naming ; //Def: %s_bisr_mem_chain_select
        }
    }
}
```

Description

A wrapper that specifies the names of the ports created when connecting the BISR chains to ports of a module of type `sub_block` or `physical_block` as specified using the `set_design_level` command. The “%s_” symbol is required but can be specified anywhere in the string. It is replaced by the power domain name followed by an underscore when the module contains power domain names other than “-”; otherwise it is replaced by an empty string. The ports are only created when the `Controller` wrapper is not specified, in which case the chains are connected to it. The ports are allowed to already be present. If the `has_functional_source` attribute on the `scan_out` port is true, the BISR chain is inserted between the net sourcing the `scan_out` port and the port itself. ICL extract will check that the chains traces back to the specified `scan_in` port. This feature is useful to support multi pass insertion flows.

Arguments

- `scan_in : port_naming ;`

A string that specifies the naming to use for the `scan_in` port. When unspecified, it defaults to the value specified in the `MemoryBisr/ChainInterface/scan_in` property inside the `DefaultsSpecification/DftSpecification` wrappers. See the `get_defaults_value` command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `scan_out : port_naming ;`

A string that specifies the naming to use for the `scan_out` port. When unspecified, it defaults to the value specified in the `MemoryBisr/ChainInterface/scan_out` property inside the

[DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `capture_shift_clock : port_naming ;`

A string that specifies the naming to use for the capture_shift_clock port. When unspecified, it defaults to the value specified in the [MemoryBisr/ChainInterface/capture_shift_clock](#) property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `shift_en : port_naming ;`

A string that specifies the naming to use for the shift_en port. When unspecified, it defaults to the value specified in the [MemoryBisr/ChainInterface/shift_en](#) property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `reset : port_naming ;`

A string that specifies the naming to use for the active low reset port. When unspecified, it defaults to the value specified in the [MemoryBisr/ChainInterface/reset](#) property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `parallel_in : port_naming ;`

A string that specifies the naming to use for the BISR chain parallel input port. This property is only effective when DftSpecification/[MemoryBisr/memory_repair_loading_method](#): from_read_buffer has been specified, otherwise it is ignored. When unspecified, it defaults to the value specified in the [MemoryBisr/ChainInterface/parallel_in](#) property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- **serial_repair_enable : port_naming ;**

A string that specifies the naming to use for the BISR chain loading enable port. This property is only effective when DftSpecification/[MemoryBisr](#)/memory_repair_loading_method: from_read_buffer has been specified, otherwise it is ignored. This port will be connected to the “SerialRepairEnable” output port of the fuse box controller. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/ChainInterface/serial_repair_enable property inside the [DefaultsSpecification](#)/[DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by the underscore when the module contains more than one power domain name; otherwise it is replaced by an empty string.

- **memory_disable : port_naming ;**

A string that specifies the naming to use for the memory_disable port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/ChainInterface/memory_disable property inside the [DefaultsSpecification](#)/[DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name otherwise, it is replaced by an empty string.

This port is only created when at least one memory instance has a Memory wrapper (see “[Tessent Core Description](#)” on page 3755) with function “bisrSerialData” and a Memory with function “select” in its Memory library description or a child block with this port is present. Such ports are identified in the ICL module as [DataInPorts](#) with the Attribute “tessent_bisr_function” set to “memory_disable”.

- **memory_chain_select : port_naming ;**

A string that specifies the naming to use for the memory_chain_select port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/ChainInterface/memory_chain_select property inside the [DefaultsSpecification](#)/[DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group, and user defaults specifications.

The string must contain the "%s_" symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

This port is only created when at least one memory instance has a Memory wrapper (see “[Tessent Core Description](#)” on page 3755) with function bisrSerialData in its Memory library description or a child block with this port is present. Such ports are identified in the

ICL module as **DataInPorts** with the **Attribute** “tessent_bisr_function” set to “memory_chain_select”.

Examples

The following example defines an interface naming convention that uses capital letters and TS_ as a prefix.

```
DftSpecification(my_design,rtl) {
    MemoryBisr {
        Interface {
            scan_in          : TS_%s_BISR_SI ;
            scan_out         : TS_%s_BISR_SO ;
            capture_shift_clock : TS_%s_BISR_CLK ;
            shift_en         : TS_%s_BISR_SHIFT_EN ;
            reset            : TS_%s_BISR_RESETB ;
            memory_disable   : TS_%s_BISR_MEM_DISABLE ;
            memory_chain_select : TS_%s_BISR_MEM_CHAIN_SEL ;
        }
    }
}
```

Controller

Specifies the creation and optionally the insertion of a BISR loader controller.

Usage

```
DftSpecification(module_name,id) {
    MemoryBisr {
        Controller {
            ijttag_host_node : node_id ;
            parent_instance : inst_name ;
            leaf_instance_name : inst_name ;
            repair_clock_connection : pin_or_port_name ;
            repair_trigger_connection : pin_or_port_name ;
            serial_repair_enable : pin_or_port_name ;//Def: auto
            programming_voltage_source : pin_or_port_name ;
            programming_voltage_port : pin_or_port_name ;
            fuse_box_location : internal | external ;
            fuse_box_interface_module : module_name ;
            max_fuse_box_programming_sessions : int | unlimited ; //Def: 1
            AdvancedOptions {
            }
            ExternalFuseBoxOptions {
            }
            PowerDomainOptions {
            }
        }
    }
}
```

Description

A wrapper that specifies the creation and optionally the insertion of a BISR loader controller. A controller needs a dedicated set of fuses associated with it and you typically need only one controller per chip. The architecture of the controller is described in the “[The BISR Controller](#)” section of the *Tessent MemoryBIST User’s Manual*.

Arguments

- `ijttag_host_node : node_id ;`

A property that specifies where the BISR controller should attached to the iJTAG network. The string `node_id` must have one of the following formats: HostScanInterface(id), Tap(id)/HostIjtag(id), ScanMux(id)/Input(1|0) or Sib(id). The specified node id must exist in the [IjtagNetwork](#) wrapper of the same [DftSpecification](#) wrapper.

When using the [create_dft_specification](#) command, the BIST controller will be attached to the Sib(sti) node. The Bisr controller is scan-testable and its `test_en` pin must be driven high during any logictest mode. The logictest modes often rely on the fact that memories were repaired prior to running the ATPG patterns when the sync write through feature is used. The `bisr_clock` ports are held low when `ltest_en` is high to make sure the BISR controller can be scan tested without disturbing the unloaded repair solution in the BISR registers.

- `parent_instance : inst_name ;`

A property that specifies the parent instance in which the memory BISR controller instance is to be inserted. When unspecified, it defaults to the top level of the design. When set to “.”, it explicitly points to the top level of the design. You specify this option if you want the memory BISR controller to be instantiated inside a different hierarchy. The parent instance must point to a unique instance; otherwise, it will be unqualified during [process_dft_specification](#).

- `leaf_instance_name : inst_name ;`

An optional property that specifies the memory BISR controller instance name explicitly. The default value is `design_name_design_id_tessent_mbisr_controller_inst`. This property is not used nor needed when the [process_dft_specification](#) command is run with the `-no_insertion` option.

- `repair_clock_connection : pin_or_port_name ;`

A property that defines a functional clock that is going to be free-running during power-up and is to be used to unload the fuse content into the BISR registers. When you point to a top level port, the connection is automatically transferred on the `from_pad` port the input pad buffer when the pad cell is properly described in your cell library.

The distributed architecture and conservative clocking methodology used for self-repair require some care in the selection of the functional repair clock that is used to apply repair during chip power-up. It is recommended that you use a functional clock of 10 MHz or less in that functional mode. Note that all other modes of operation for the BISR controller used for manufacturing test use the TAP clock (TCK), which is also 10 MHz, by default. Using such a low frequency simplifies timing closure. At 10 MHz, self-repair will take approximately 1 ms to execute for a BISR chain with 5000 bits, which is sufficient to repair hundreds of memories. If this time needs to be shortened, you can use a faster functional clock. However, you must consider the following factors when making a decision because they might limit the maximum achievable frequency:

- Clock balancing of the BISR chain

The BISR chain is distributed throughout the chip, and some segments of the BISR chain might be part of pre-designed circuit blocks. Balancing the BISR clock to achieve a higher speed of operation might be difficult.

- Clock edge uncertainty

Retiming registers are inserted between BISR registers so that balancing the BISR clock is not necessary. However, these retiming registers might limit the maximum achievable frequency due to the uncertain time of occurrence of the falling BISR clock edge with respect to its rising edge.

- Distribution of the BISR registers

The BISR chain is distributed throughout the chip, and some segments of the BISR chain might be part of pre-designed circuit blocks. Long wires between BISR registers might limit the maximum achievable frequency. Note that the routing of the

BISR chain is not optimized based on placement unless have read in a DEF file using the [read_def](#) command prior to running the [check_design_rules](#) command.

- **repair_trigger_connection : *pin_or_port_name* ;**

A property that specifies the functional signal that resets the BISR controller and BISR registers and initiates the memory repair. This signal is typically derived from a power-on detector or the power management unit of your device. When you point to a top level port, the connection is automatically transferred on the *from_pad* port of the input pad buffer when the pad cell is properly described in your cell library.

- **serial_repair_enable : *pin_or_port_name* ;**

A property that specifies the input that selects between serial or parallel BISR chain loading. The specified pin or port will be connected to the “SerialRepairEnable” input port of the fuse box controller. When driven with a “0” the BISR chains are loaded in parallel from the read buffer of the fuse box. When driven with a “1”, the fusebox controller initializes the BISR chains serially. This property is required when the [MemoryBisr/memory_repair_loading_method](#) parameter is set to “from_read_buffer”, otherwise it is ignored.

- **programming_voltage_source : *pin_or_port_name* ;**

A property that defines the top-level port or internal voltage source that supplies the programming voltage for the fuse box programming. If the programming voltage is generated using an on-chip charge pump, then specify the voltage output pin of the charge pump. This property must be specified when the *fuse_box_location* property is set to internal. This property is used to make the connection between the specified programming voltage source and the fuse box programming voltage pin. You do not need to specify this property when the *fuse_box_location* property is set to external because the programming voltage source should already be connected to the fuse box interface in the pre-dft inserted design. Refer to “[Connecting the BISR Controller to an External Fuse Box](#)” for further information about external fuse box implementation.

- **programming_voltage_port : *port_name* ;**

An optional property that defines the top-level port that supplies the programming voltage. The voltage port must be identified if it needs to be driven to a specific programming value when the fuses are being programmed. You do not need to specify this property when the *programming_voltage_source* is specified and references a top-level port or when the voltage is generated from an internal charge pump.

- **fuse_box_location : internal | external ;**

A property that specifies if the fuse box is to be instantiated inside the BISR controller, or if it already exists in the design and connections should be made to and from it. It is typically external when the up address range of the fuse box is used by other functional or test modes. Refer to the [ExternalFuseBoxOptions](#) wrapper if your fuse box is external to the controller.

When not specified, this property defaults to the [MemoryBisr/fuse_box_location](#) property found inside the [DefaultsSpecification/DftSpecification](#) wrapper.

- `fuse_box_interface_module : module_name ;`

A property that specifies the module name of the fuse box to instantiate in the design. The module name specified with this property will override any that are inferred from `tcd_fusebox` libraries loaded in the design. If the fuse box module is already instantiated into the design, its associated [FuseBoxInterface](#) wrapper is automatically searched and loaded. See the [set_module_matching_options](#) and the “[set_design_sources -format tcd_fusebox](#)” command descriptions for more information about the searching mechanism.

When the `fuse_box_location` property is set to internal and only one `tcd_fusebox` file is present, the `fuse_box_interface_module` property is inferred from the `tcd_fusebox` if the `fuse_box_interface_module` property is not specified. If multiple `tcd_fusebox` library files are present, then it is required to specify the `fuse_box_interface_module` property in this wrapper.

When the `fuse_box_location` property is set to external and `fuse_box_interface_module` is not specified, the `fuse_box_interface_module` will be inferred from the specified [ExternalFuseBoxOptions/design_instance](#) property. If neither the design instance or `fuse_box_interface_module` properties are specified, and only a single `tcd_fusebox` exists in the design, the `fuse_box_interface_module` will be inferred from this `tcd_fusebox`. If multiple `tcd_fusebox` files are present, an error will result and the user will be required to specify which `fuse_box_interface_module` to use.

Refer to the [FuseBoxInterface](#) wrapper description and “[Connecting the BISR Controller to an External Fuse Box](#)” topic in the *Tessent MemoryBIST User’s Manual* for further information on `tcd_fusebox` usage and DftSpecification requirements.

- `max_fuse_box_programming_sessions : int | unlimited ;`

A property that enables hard incremental repair and specifies how many fuse box programming sessions are to be supported by the controller. You need more than one programming session if you need to test the memories at different times and conditions, and if you want to be able to program the repair fuses after each repair analysis. This is useful for increasing the yield of circuits exhibiting parametric defects under certain process, voltage or temperature conditions. This approach avoids the need to save the repair information of the chip for each intermediate repair solution, and only program the fuse once the cumulative failure map is known. Refer to the [Tessent MemoryBIST User’s Manual](#) for a detailed description of this feature. The hardware cost associated with each fuse box programming session is a register that is as wide as the fuse box address bus, plus an extra flop and some logic gates.

When specifying an integer value, the repair information is compressed before being stored in the fuse box. The number of fuses required to store the repair information is proportional to the integer value.

When specified as ‘unlimited’, the fuse box compression hardware is disabled inside the fuse box controller. This enables the repair solution from the BISR chains to be stored uncompressed inside the fuse box. New repair solutions can be written directly inside the fuse box without affecting the repair solution from previous repair sessions. This allows an unlimited number of self_fuse_box_program autonomous mode programming sessions.

Disabling the compression hardware is also useful in the rare case where more than 50% of the repair registers are expected to be used. The compression algorithm becomes inefficient in that case and might require more fuses than there are bits in the BISR chain.

Note

 When fuse box compression is disabled through the ‘unlimited’ setting, the fuse box size must be greater than or equal to the total number of bits in the BISR chains.

When not specified, this property defaults to the [MemoryBISR](#)/
max_fuse_box_programming_sessions property found inside the [DefaultsSpecification](#)/
[DftSpecification](#) wrapper.

Examples

The following example shows the specification of the controller wrapper. The ijttag_host_node property is linked to Sib(bisr) node. The repair_clock_connection property points to a port of the device. The cell library information will be automatically used to map the connection on the other side of the input pad buffer. The repair_trigger_connection property is used to specify the connection to the power management unit. Because the fuse box is internal, FuseBoxInterface(my_fuse_box_internal) wrapper (see “[Tessent Core Description](#)” on page 3755) is required to have been pre-loaded into memory using the [read_core_descriptions](#) command.

```
DftSpecification(my_design,rtl) {
    IJTAGNetwork {
        HostScanInterface(ijttag) {
            Sib(sti) {
                Sib(bisr) {
                }
            }
        }
    }
    MemoryBISR {
        Controller {
            ijttag_host_node : Sid(bisr) ;
            repair_clock_connection : Clk10Mhz ;
            repair_trigger_connection : pmu/repair_en ;
            programming_voltage_source : VPP ;
            fuse_box_location : internal ;
            fuse_box_interface_module : my_fuse_box_interface ;
        }
    }
}
```

AdvancedOptions

Specifies advanced options for the BISR controller.

Usage

```
DftSpecification(module_name, id) {
    MemoryBISR {
        Controller {
            AdvancedOptions {
                FuseBoxOptions {
                    write_duration : time | 8us ;
                    read_word_size : int | auto ;
                    number_of_fuses_for_repair : int | auto ;
                    align_access_en_with_address : on | off | auto ;
                    programming_method : buffered | unbuffered | auto ;
                }
                repair_word_size : int | auto ;
                max_bisr_chain_length : int | auto ;
                zero_counter_bits : int | auto ;
                bisr_done_connection : pin_name ;
                bisr_pass_connection : pin_name ;
                power_up_chain_select : internal | external ;
            }
        }
    }
}
```

Description

A wrapper that specifies advanced options for the BISR controller. You do not need to specify any of the properties with an auto default value when you have a FuseBoxInterface wrapper (see “[Tessent Core Description](#)” on page 3755) associated with the specified fuse_box_module_name property. The two connection properties are typically used to connect the status signal to the power management unit when different portions of the chip are powered up and down during the normal operation of a device, and those portions have repairable memories that need to be repaired each time the region is turned back on. Refer to the [Tessent MemoryBIST User’s Manual](#) for detailed information about power domains and their impact on memory BISR.

Arguments

- FuseBoxOptions/write_duration : *time* | 8us ;
 A property that specifies the time it takes to program a fuse. When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, the default value of “8us” is used. The write duration of the fuse box is the value found in the datasheet of the fuse box.
- FuseBoxOptions/read_word_size : *int* | auto ;
 A property that specifies how many fuses are read in parallel. The fuse box interface module is designed to not reread the fuse box if the next read address it is getting is part of the same word that it just read. This information is needed to properly compute how long it takes to

read the fuse values and estimate the worst case repair time needed when all fuses need to be read.

When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, the default value of 1 is used. It is ok to specify a read_word_size value that is actually smaller than the true word size of the fuse box that was read as it will only makes the calculated worst case run time for the power up emulation mode longer. This property does not affect the controller hardware and the true worst case time it takes to power up a domain. However, it is important to not specify a read_word_size that is larger than reality as the power up emulation pattern would be generated with a wait loop that is too short.

- FuseBoxOptions/number_of_fuses_for_repair : *int* | auto ;

A property that defines the number of fuses available for repair in the fuse box. You need to reserve the bottom part of the address space for repair but you may use the upper part for other purposes. When the value is auto, the number of fuses for repair is set to the value of number_of_fuses specified in the Core/FuseBoxInterface wrapper. If no FuseBoxInterface description is available, the value is computed as 2 to the power of the number of address ports specified with the ExternalFuseBoxOptions/ConnectionOverrides/address property.

- FuseBoxOptions/align_access_en_with_address : on | off | auto ;

A property that specifies if the pulse triggering a fuse box access and generated by the BISR controller is expected to be aligned with the address to be read or written or if it is one cycle ahead. The value “on” means that the fuse box interface can only accept a pulse that is aligned with the address. The value “off” means that the fuse box interface can also accept a pulse occurring one cycle ahead of the address allowing a slightly faster fuse box access. For example, a read access can take as little as two clock cycles instead of three for some fuse box interfaces. When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, it defaults to on if fuse_box_location is external, and to off if fuse_box_location is internal.

- programming_method : buffered | unbuffered | auto ;

A property that specifies that it is not allowed to program individual fuses at randomly specified addresses. In some fuseboxes, individual fuse bits cannot be addressed directly; all fuse bits must be read or programmed as a group. When this property is set to buffered, a signal called programFB is generated by the BISR controller and connected to the fuse box interface. This signal is used to initiate final fuse box programming. Extra steps are also added to the patterns to transfer and program the fuses during the “self_fuse_box_program” autonomous operation and in the “program” and “read” fuse box access modes. When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, it defaults to “unbuffered”.

- repair_word_size : *int* | auto ;

A property that specifies the repair word size which is the number of bits read from the fuse box as soon as one of the bits in the word is a one. Refer to the compression algorithm description in the [Tessent MemoryBIST User’s Manual](#) for more information.

When set to auto, this property is automatically set to match the largest repair word size of all your repairable memories providing the most effective encoding capacity.

- max_bisr_chain_length : *int | auto* ;

A property that specifies the maximum length a BISR chain could have. Unless you are creating a BISR controller without having all BISR chains visible, you do not need to specify this property because the default auto value implies that the controller is built to be capable of loading BISR chains that are as much as four times larger than they currently are, allowing for any number of last minute additions.

- zero_counter_bits : *int | auto* ;

A property that specifies how many bits the zero counter of the encoder must have. This property should never need to be specified as it is automatically set to the log2 of the specified or inferred max_bisr_chain_length property. Refer to the compression algorithm description in the [Tessent MemoryBIST User's Manual](#) for more information about the zero counter.

- bisr_done_connection : *pin_name* ;

A property that specifies a connection to be made from the done status port of the BISR controller to the power management unit. This done status port indicates that all enabled power domains are programmed. See the done_to_pmu_connection property inside the [PowerDomainOptions](#) wrapper to know how to connect each power domain-specific done status port to the power management unit.

- bisr_pass_connection : *pin_name* ;

A property that specifies a connection to be made from the pass status port of the BISR controller to the power management unit. This pass status port indicates that no errors happened while reading the fuse and loading the enabled BISR chains.

- power_up_chain_select : *internal | external* ;

This property changes the default BISR chain select output on the fuse box controller when performing a power-up or a calculate_bisr_chain_length pattern. This property is used for designs that contain memories with serial repair interfaces that do not perform a complete reset of their internal BISR chain during an asynchronous reset from the fuse box controller. Certain TSMC repairable memories implement such a BISR reset mechanism.

When power_up_chain_select is set to internal, the fuse box controller selects the internal BISR chains when performing the chain length calculation. However, if the memory's internal BISR chain does not reset completely, this will cause the BISR chain length calculation to fail. It is recommended to use the "internal" chain when possible in order to verify the internal chains every time the chip is powered up.

When set to external, the external BISR register is used to calculate the BISR chain length during power up. This register is guaranteed to reset completely and ensures that the BISR chain length calculation succeeds.

Examples

The following example connects the done and pass status ports to pins of the power management unit.

```
DftSpecification(my_design,rtl) {
    MemoryBisr {
        Controller {
            AdvancedOptions {
                bisr_done_connection      : pmu/global_bisr_done ;
                bisr_pass_connection      : pmu/global_bisr_pass ;
            }
        }
    }
}
```

ExternalFuseBoxOptions

Specifies options relative to an external fuse box and its interface.

Usage

```
DftSpecification(module_name, id) {
    MemoryBisr {
        Controller {
            ExternalFuseBoxOptions {
                design_instance      : inst_name ;
                multiplexing         : on | off | auto ;
                ConnectionOverrides {
                    bisr_en          : pin_name, ... ;
                    clock             : pin_name ;
                    select            : pin_name ;
                    reset             : pin_name ;
                    access_en         : pin_name ;
                    write_en          : pin_name ;
                    write_duration_count : pin_name ;
                    read_buffer_select : pin_name ;
                    read_buffer_output : pin_name ;
                    write_buffer_transfer : pin_name ;
                    address           : pin_name ;
                    done              : pin_name ;
                    read_data          : pin_name ;
                }
            }
        }
    }
}
```

Description

A wrapper that specifies options relative to the external fuse box and its interface. This wrapper is only used when the `fuse_box_location` property in the `DftSpecification/MemoryBisr/Controller` wrapper is set to external. The `ConnectionOverrides` wrapper is used to define connections between the external fuse box and BISR controller. These connections can also be specified in the `tcd_fusebox FuseBoxInterface` module library. When defining connections in the `ConnectionOverrides` wrapper, it is mandatory to specify the following properties:

- `done`
- `read_data`
- `write_en`
- `select`
- `access_en`
- `address`
- `write_duration_count`

Refer to the `fuse_box_location` and `fuse_box_interface_module` property descriptions, the [FuseBoxInterface](#) wrapper description and the “[Connecting the BISR Controller to an External Fuse Box](#)” topic in the *Tessent MemoryBIST User’s Manual* for further details.

Arguments

- `design_instance : inst_name ;`

A property that is only specified when the `DftSpecification/MemoryBISR/Controller` `fuse_box_interface_module` property is not specified. The `fuse_box_interface_module` will be inferred from the `design_instance` property. If neither the `design_instance` or `fuse_box_interface_module` properties are specified, and only a single `tcd_fusebox` exists in the design, the `fuse_box_interface_module` will be inferred from this `tcd_fusebox`. If multiple `tcd_fusebox` files are present, an error will result and the user will be required to specify which `fuse_box_interface_module` to use.

- `multiplexing : on | off | auto ;`

A property that specifies if the connection to the external fuse box should be multiplexed or not. The connection must be multiplexed if it is also used by some other circuitry. If it is multiplexed, a 2-to-1 multiplexer local to the connection end point is inserted and the select is driven by the `bisr_en` signal. When the property is set to `auto`, the multiplexer is only inserted if the `has_functional_source` Pin attribute of the specified `pin_name` in the connection is set to true. See “[Pin](#)” on page 3094.

- `ConnectionOverrides/bisr_en : pin_name ;`

A property that specifies one or many connections to create from the `bisr_en` port to any pins in the circuit. This signal is typically used as a select of multiplexers when the fuse box is shared with other circuitry and the multiplexer is already inside the interface.

- `ConnectionOverrides/clock : pin_name ;`

A property that specifies where to connect the clock signal for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the clock connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the clock signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- `ConnectionOverrides/select : pin_name ;`

A property that specifies where to connect the select signal for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the select connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the select signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- `ConnectionOverrides/reset : pin_name ;`

A property that specifies where to connect the reset signal for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the reset connection needs to be made to any

place other than the ports of the fuse box interface module. For a description of the reset signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- ConnectionOverrides/access_en : *pin_name* ;

A property that specifies where to connect the access_en signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the access_enable connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the access_enable signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- ConnectionOverrides/write_en : *pin_name* ;

A property that specifies where to connect the write_en signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the write_enable connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the write_enable signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- ConnectionOverrides/write_duration_count : *pin_name* ;

A property that specifies where to connect the write_duration_count signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the write_duration_count connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the write_duration_count signals on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

If the write_duration_count pins are a series of numbered scalar ports, you can use the %<integer>d[msb:lsb] symbol to specify them. For example, wcount%2d[31:0] will connect to the scalar ports called wcount31 to wcount00.

- ConnectionOverrides/read_buffer_select : *pin_name* ;

A property that specifies where to connect the read_buffer_select signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the read_buffer_select connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the read_buffer_select signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

- ConnectionOverrides/read_buffer_output : *pin_name* ;

A property that specifies where to connect the read_buffer_output signals for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the read_buffer_output connections need to be made to any place other than the ports of the fuse box interface module. For a description of the read_buffer_output signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

If the `read_buffer_output` pins are a series of numbered scalar ports, you can use the `%<integer>d[msb:lsb]` symbol to specify them. For example `RB%2d[12:0]` will connect to the scalar ports called RB12 to RB00.

- `ConnectionOverrides/write_buffer_transfer : pin_name ;`
A property that specifies where to connect the `write_buffer_transfer` signal for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the `write_buffer_enable` connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the `write_buffer_transfer` signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.
- `ConnectionOverrides/address : pin_name ;`
A property that specifies where to connect the address signals for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the address connection needs to be made to any place other than the port of the fuse box interface module. For a description of the address signals on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.
If the address pins are a series of numbered scalar ports, you can use the `%<integer>d[msb:lsb]` symbol to specify them. For example `add%2d[13:0]` will connect to the scalar ports called add13 to add00.
- `ConnectionOverrides/done : pin_name ;`
A property that specifies where to connect the `done` signal from the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the `done` connection needs to be made from any other place than the ports of the fuse box interface module. For a description of the `done` signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.
- `ConnectionOverrides/read_data : pin_name ;`
A property that specifies where to connect the `read_data` signal for the fuse box interface. This property is used when you do not have a `FuseBoxInterface` wrapper associated with a `fuse_box_module_interface` property, or if the `read_value` connection needs to be made from any place other than the port of the fuse box interface module. For a description of the `read_value` signal on the fuse box interface, refer to the [FuseBoxInterface](#) wrapper description.

Examples

The following examples uses the `design_instance` property to specify which instance of the fuse box module is to be used by the BISR controller. This is only needed in the rare case that the design contains more than one instance of the module specified by the `fuse_box_interface_module` property. The clock connection is overridden and connects to the input of a clock buffer instead of to the pin of the fuse box interface module.

```
DftSpecification(my_design,rtl) {
    MemoryBisr {
        Controller {
            ExternalFuseBoxOptions {
                design_instance : fuse_box_inst1 ;
                ConnectionOverrides {
                    clock          : clock_buf/A ;
                }
            }
        }
    }
}
```

PowerDomainOptions

Specifies options specific to the power domains.

Usage

```
DftSpecification(module_name, id) {
    MemoryBisr {
        Controller {
            PowerDomainOptions {
                power_domain_priority_order : power_domain_name, ... ;
                PowerDomainName(power_domain_name) {
                    enable_from_pmu_connection : pin_or_net_name ;
                    busy_to_pmu_connection : pin_or_net_name ;
                    done_to_pmu_connection : pin_or_net_name ;
                    ChainConnectionOverride {
                        to_scan_in : port_or_pin_name ;
                        from_scan_out : port_or_pin_name ;
                        capture_shift_clock : port_or_pin_name ;
                        shift_en : port_or_pin_name ;
                        reset : port_or_pin_name ;
                        memory_disable : port_or_pin_name ;
                        memory_chain_select : port_or_pin_name ;
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that specifies options specific to the power domains. The `power_domain_priority_order` property is used to specify in which order the power domains are prioritized during the power up sequence.

You should order the list such that the power domains that must be programmed first are listed first. You use the `PowerDomainName` wrapper to specify connections between the controller and the power management unit. The `power_domain_name` used to identify the `PowerDomainName` wrappers must match the value of the `bisr_power_domain_name` attribute associated with memory and BISR segment instances. For more details about the usage of the `bisr_power_domain_name` attribute, see the description of the [MemoryBisr](#) wrapper.

Arguments

- `power_domain_priority_order : power_domain_name, ... ;`

A property that lists the power domain names in the order they must be programmed during power up. The power domains that must be powered-up faster than the others should be specified at the beginning of the list. If you are not using the `-no_insertion` mode of the [process_dft_specification](#) command and a memory and BISR segment instance have a `bisr_power_domain_name` attribute value that is not found in the list, an error is generated. An error is also generated if the list includes a `power_domain_name` that is not associated with any memory or BISR segment instances, unless the `ChainConnectionOverride` wrapper

is specified in the corresponding PowerDomainName wrapper. See [Example 2](#) for an example usage of the ChainConnectionOverride wrapper.

- PowerDomainName/enable_from_pmu_connection : *pin_or_net_name* ;
A property that specifies a connection from the power management unit to the power domain-specific enable signal. Refer to the [Tessent MemoryBIST User's Manual](#) for more information about the signals between the power management unit and the BISR controller.
- PowerDomainName/busy_to_pmu_connection : *pin_or_net_name* ;
A property that specifies a connection to the power management unit from the power domain-specific BISR busy signal. Refer to the [Tessent MemoryBIST User's Manual](#) for more information about the signals between the power management unit and the BISR controller.
- PowerDomainName/done_to_pmu_connection : *pin_or_net_name* ;
A property that specifies a connection to the power management unit from the power domain-specific BISR done signal. Refer to the [Tessent MemoryBIST User's Manual](#) for more information about the signals between the power management unit and the BISR controller.
- PowerDomainName/ChainConnectionOverride/to_scan_in : *port_or_pin_name* ;
A property that overrides the scan_in connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See the [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.
- PowerDomainName/ChainConnectionOverride/from_scan_out : *port_or_pin_name* ;
A property that overrides the scan_out connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See the [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.
- PowerDomainName/ChainConnectionOverride/capture_shift_clock : *port_or_pin_name* ;
A property that overrides the capture_shift_clock connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/shift_en : *port_or_pin_name* ;
A property that overrides the shift_en connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.
- PowerDomainName/ChainConnectionOverride/reset : *port_or_pin_name* ;
A property that overrides the reset connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.
- PowerDomainName/ChainConnectionOverride/memory_disable : *port_or_pin_name* ;
A property that overrides the memory_disable connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See [Example 2](#) below for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time. The specification of the memory_disable connection overrides remains optional because this is an optional connection that is only used if your memories have serial repair access as described in the *Memory BISR User's Manual*.
- PowerDomainName/ChainConnectionOverride/memory_chain_select : *port_or_pin_name* ;
A property that overrides the memory chain select connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. See [Example 2](#) for an example usage. You must override all connections for a given power domain name as soon as you override one. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

Examples

Example 1

The following example defines the priority order for the power domain names A, B, and C where power domain A has highest priority and C has the lowest priority. The connections per power domain are also specified.

```
DftSpecification(my_design,rtl) {
    MemoryBisr {
        Controller {
            PowerDomainOptions {
                power_domain_priority_order : A,B,C ;
                PowerDomainName(A) {
                    enable_from_pmu_connection : pmu/bisr_en_A ;
                    busy_to_pmu_connection : pmu/bisr_busy_A ;
                    done_to_pmu_connection : pmu/bisr_done_A ;
                }
                PowerDomainName(B) {
                    enable_from_pmu_connection : pmu/bisr_en_B ;
                    busy_to_pmu_connection : pmu/bisr_busy_B ;
                    done_to_pmu_connection : pmu/bisr_done_B ;
                }
                PowerDomainName(C) {
                    enable_from_pmu_connection : pmu/bisr_en_C ;
                    busy_to_pmu_connection : pmu/bisr_busy_C ;
                    done_to_pmu_connection : pmu/bisr_done_C ;
                }
            }
        }
    }
}
```

Example 2

The following example defines explicit connections for the chains associated with the power domain names. This syntax is used when constructing and inserting a BISR controller into a design in which the chains are not visible. In this example, the BISR controller is inserted into a module that will be used in many designs. The module already contains the power management unit and an instance of the fuse box interface module called fuse_2k module. The list of power domain names and the max chain length and repair word size values are specified because their values cannot be extracted from the chains because they do not exist yet.

```

DftSpecification(my_design,rtl) {
    MemoryBisr {
        Controller {
            ijttag_host_node : Sib(bisr) ;
            repair_clock_connection : bisr_clk ;
            repair_trigger_connection : pmu/bisr_trigger ;
            fuse_box_location : external ;
            fuse_box_interface_module : fuse_2k ;
        }
        repair_word_size : 8 ;
        max_bisr_chain_length : 4000 ;
        bisr_done_connection : pmu/bisr_done ;
        bisr_pass_connection : pmu/bisr_pass ;
    }
    PowerDomainOptions {
        power_domain_priority_order : A,B,C ;
        PowerDomainName(A) {
            enable_from_pmu_connection : pmu/bisr_en_A ;
            busy_to_pmu_connection : pmu/bisr_busy_A ;
            done_to_pmu_connection : pmu/bisr_done_A ;
            ChainConnectionOverride {
                to_scan_in : A_bisr_si ;
                from_scan_in : A_bisr_so ;
                capture_shift_clock : A_bisr_clk ;
                shift_en : A_bisr_se ;
                reset : A_bisr_rst ;
                memory_disable : A_bisr_md ;
                memory_chain_select : A_bisr_csel ;
            }
        }
        PowerDomainName(B) {
            enable_from_pmu_connection : pmu/bisr_en_B ;
            busy_to_pmu_connection : pmu/bisr_busy_B ;
            done_to_pmu_connection : pmu/bisr_done_B ;
            ChainConnectionOverride {
                to_scan_in : B_bisr_si ;
                from_scan_in : B_bisr_so ;
                capture_shift_clock : B_bisr_clk ;
                shift_en : B_bisr_se ;
                reset : B_bisr_rst ;
                memory_disable : B_bisr_md ;
                memory_chain_select : B_bisr_csel ;
            }
        }
        PowerDomainName(C) {
            enable_from_pmu_connection : pmu/bisr_en_C ;
            busy_to_pmu_connection : pmu/bisr_busy_C ;
            done_to_pmu_connection : pmu/bisr_done_C ;
            ChainConnectionOverride {
                to_scan_in : C_bisr_si ;
                from_scan_in : C_bisr_so ;
                capture_shift_clock : C_bisr_clk ;
                shift_en : C_bisr_se ;
                reset : C_bisr_rst ;
                memory_disable : C_bisr_md ;
            }
        }
    }
}

```

}

MemoryBist

Specifies the creation and the optional insertion of memory BIST controllers in a design.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        ijtag_host_node : node_id ;
        Controller(id)
    }
    ReusedController(id) {
    }
    BistAccessPort {
    }
    TCKInjectionPoints {
    }
}
```

Description

A wrapper that specifies the creation and the optional insertion of memory BIST controllers in a design.

The presence of the MemoryBist wrapper within the DftSpecification adds one or more controllers to the current design. Refer to the [Tessent MemoryBIST User's Manual](#) to get started with the MemoryBist insertion flow.

Arguments

- *ijtag_host_node* : *node_id* ;
A required property that specifies the ScanInterface that is hosting the memory BIST controllers. The *node_id* is of the form “Sib(*id*)” when the **DftSpecification** is created using the [create_dft_specification](#) command. However, the *node_id* is allowed to point to any valid host scan interface: [Tap/HostIjtag](#), [Sib\(*id*\)](#), or [HostScanInterface/Interface\(*id*\)](#).

Controller

A wrapper that defines a new memory BIST controller. The *id* value uniquely identifies the controller. The *id* value is reflected in the module names associated with that controller.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        Controller(id) {
            parent_instance           : inst_path ;
            leaf_instance_name        : inst_name ;
            clock_domain_label        : label ;
            clock_period               : time ;
            AdvancedOptions           { // *DefSpec
            }
            AlgorithmResourceOptions   { // *DefSpec
            }
            RepairOptions              { // *DefSpec
            }
            DiagnosisOptions           { // *DefSpec
            }
            Step                      { // Mutually exclusive with
            }                           // MemoryCluster wrapper
            MemoryCluster(cluster_id) { // Mutually exclusive with
            }                           // Step wrapper
        }
    }
}
```

Description

A wrapper that defines a new memory BIST controller. The *id* value uniquely identifies the controller. The *id* value is reflected in the module names associated with that controller.

Arguments

- parent_instance : *inst_path* ;

An optional property that specifies where the memory BIST controller is instantiated in the design. When set to “.”, it points to the top level of the design. When unspecified, it defaults to a null string that is interpreted as the common instance ancestor of all memories tested by the controller. See the description of the [get_common_parent_instance](#) command to see how the common ancestor is computed.

This property is not used nor needed when the [process_dft_specification](#) command is run with the -no_insertion option.

- leaf_instance_name : *inst_name* ;

An optional property that specifies the memory BIST controller instance name explicitly. By default, an instance name is automatically generated by appending the “_inst” string to the controller module name, which itself defaults to the following:

design_name_design_id_tessent_mbist_controller_id_controller

This property is not used nor needed when the [process_dft_specification](#) command is run with the -no_insertion option.

- `clock_domain_label : label ;`

A property that specifies the label (see [add_clocks](#)) of the clock domain used by the memory BIST controller and all the associated memories. Any memory tested by the controller that has a clock port sourced by a different clock domain will have a multiplexer added in its interface module to inject the clock used by the controller when running memory BIST.

This property is not used nor needed when the [process_dft_specification](#) command is run with the -no_insertion option and the design is not present. When running in -no_insertion mode without the design loaded in memory, specify the `clock_period` property instead. This is needed to correctly create the synthesis script.

- `clock_period : time ;`

A property that is only used when [process_dft_specification](#) is run with the -no_insertion option and the design with the memory instances is not loaded into the tool. In this case, the `clock_domain_label` property cannot be specified and instead the `clock_period` property is used. When the design is present, the clock period used in the synthesis script is extracted from the period of the clock referred to by the `clock_domain_label` property. In -no_insertion mode, there may be no design present with defined clocks to refer to. In this case, you use the `clock_period` property to specify at which rate the given controller is meant to operate.

Also note that when the `clock_domain_label` property is not specified, the information needed to control the insertion of the clock multiplexers inside the memory interface modules is missing. You are therefore responsible for adding those multiplexers where needed when you insert the controller inside your design such that all memory clocks are driven by the clock used by the controller during memory BIST.

AdvancedOptions

Specifies advanced options for the memory BIST controller.

Usage

```
DftSpecification(module_name,id) {
    MemoryBist {
        Controller(id) {
            AdvancedOptions {
                algorithm : algo_name ; // *DefSpec
                // default: from_library
                operation_set : opset_name ;
                // default: from_library
                extra_algorithms : algo_name, ... ;
                extra_operation_sets : opset_name, ... ;
                functional_debug_mode : on | off ; // *DefSpec
                pipeline_controller_outputs : on | off ; // *DefSpec
                observation_xor_size : 1..8 | off ; // default:3
                selective_parallel_memory_test : on | off ; // *DefSpec
                shared_comparators_per_go_id : int | all ;
                //default: 1 *DefSpec
                use_multicycle_paths : on | off ; // *DefSpec
                gate_bist_clock_in_functional_mode : off |
                    with_clock_gating_cell | with_clock_mux; // *DefSpec
                min_misr_segment_bits : 24 | 256 ; // *DefSpec
            }
        }
    }
}
```

Description

A wrapper that specifies advanced options for the memory BIST controller.

Arguments

- `algorithm : algo_name ;`

An optional property that specifies the name of an algorithm. This value serves as the default value for the [Step](#)/algorithm property when it is not specified in the [Step](#) wrapper. When unspecified, it defaults to “from_library”. See the [Step](#)/algorithm property description to understand the meaning of the default value.

The allowed list of algorithm names already existing in the tool is LVAddressInterconnect, LVBitSurroundDisturb, LVCheckerboard1X1, LVCheckerboard4X4, LVColumnBar, LVDataInterconnect, LVGalColumn, LVGalPat, LVGalRow, LVMarchCMinus, LVMarchLA, LVMarchX, LVMarchY, LVMasest, LVRowBar, LVWalkingPat, ReadOnly, March, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, and SMarchCHKBbcd. Those algorithm are documented in the “[Mentor Graphics Library of Algorithms](#)” section of the *Tessent MemoryBIST User’s and Reference Manual*.

You can also define and name your own algorithm using the Algorithm wrapper.

- `operation_set : string ;`

An optional property that specifies the name of an operation set. This value serves as the default value for the [Step/operation_set](#) property when it is not specified in the [Step](#) wrapper. When unspecified, it defaults to “from_library”. See the [Step/operation_set](#) property description to understand the meaning of the default value.

The allowed list of operation set names already existing in the tool is Async, AsyncWR, Sync, SyncWR, SyncWRvcd, clockedAsync, clockedAsyncWR, and rom. Those operation sets are documented in the [Built-In OperationSet](#) section of the *Tessent MemoryBIST User’s and Reference Manual*.

You can also define and name your own operation set using the [OperationSet](#) wrapper.

- `extra_algorithms: algo_name, ... ;`

An optional property that specifies the names of additional algorithms that are to be built into the controller such that they can later be selected at run time. The algorithms can be explicitly listed or defined using a regular expression.

The allowed list of algorithm names already existing in the tool is LVAddressInterconnect, LVBitSurroundDisturb, LVCheckerboard1X1, LVCheckerboard4X4, LVColumnBar, LVDataInterconnect, LVGalColumn, LVGalPat, LVGalRow, LVMarchCMinus, LVMarchLA, LVMarchX, LVMarchY, LVMases, LVRowBar, LVWalkingPat, ReadOnly, March, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, and SMarchCHKBvcd. Those algorithm are documented in the [Mentor Graphics Library of Algorithms](#) section of the *Tessent MemoryBIST User’s and Reference Manual*.

- `extra_operation_sets: opset_name, ... ;`

An optional property that specifies the names of additional operation sets that can be built into the controller such that they can later be selected at run time. The operation set names can be explicitly listed or defined using a regular expression.

The allowed list of operation set names already existing in the tool is Async, AsyncWR, Sync, SyncWR, SyncWRvcd, clockedAsync, clockedAsyncWR, and rom. Those operation sets are documented in the [Built-In OperationSet](#) section of the *Tessent MemoryBIST User’s and Reference Manual*.

You can also define and name your own operation set using the [OperationSet](#) wrapper.

- `functional_debug_mode: on | off ;`

An optional property that specifies that the necessary setup chain muxing and controller ports are generated to allow functional system debug. Provides read and write access to any BISTable memory with a negligible impact on area. For more information, see the [“Functional Debug Memory Access” Appendix](#) in *Tessent MemoryBIST User’s Manual for Use with Tessent Shell*.

- `pipeline_controller_outputs : on | off ;`

A property that specifies the insertion of one pipeline stage on all BIST outputs from the memory BIST controller. The pipeline stage is added for all controller steps and memories

under test. The pipeline flip-flops are instantiated in the controller module. This property facilitates timing closure on paths from the controller.

- `observation_xor_size : 1..8 | off ;`

A property that specifies the number of signals combined in an XOR tree for observing the address and control ports of the memories in a memoryBIST interface during logic test. If not specified, a default value of 3 is used. When the off value is specified, no observation logic for logic test is inserted into the interface modules.

- `selective_parallel_memory_test: on | off ;`

A property that specifies whether or not specific memories are to be enabled during memory BIST. The compare status of any disabled memories will remain at the pass status throughout the test and will not contribute to the global pass/fail status. In addition, disabled memory inputs with port function Select and WriteEnable are forced to their inactive value. The memories to be enabled for memory BIST are selected at runtime using the `enable_memory_list` property in the PatternsSpecification/Patterns/TestStep/MemoryBist/Controller/[AdvancedOptions](#) wrapper.

- `shared_comparators_per_go_id : int | all ;`

A property that specifies the number of comparators to share a given go_id register located in the controller. You can use this feature to manage the area overhead by reducing the number of registers used to log the comparator status. However, using a value other than 1 complicates and lengthens diagnosis with bit level resolution as the test needs to be repeated for as many times as there are comparators sharing a common go_id register.

The value should be set to 1 to enable reading complete memory words in a single read operation. Also, the MemoryBist/Controller/[AlgorithmResourceOptions](#)/data_register_bits property should be set to an appropriate value based on the complexity of the patterns to be written to memory. For complete control of the write data, this property should be set to the number of bits of the widest memory tested by the controller.

Specifying the value “all” infers that there will be one go_id register for all comparators located in the controller.

- `use_multicycle_paths : on | off ;`

A property that specifies whether or not multi-cycle paths are declared in the generated SDC and STA scripts. Omitting those greatly simplifies the SDC constraints but limits the speed at which the controller can operate as all timing paths needs to be closed as if they were all single-cycle paths when the timing exceptions are not declared.

- `gate_bist_clock_in_functional_mode : off | with_clock_gating_cell | with_clock_mux;`

An optional property that disables the clock input to the memory BIST controller and interfaces during the functional mode of operation. This will reduce clocking activity in the chip, saving power. The value “with_clock_mux” inserts a multiplexer with the 0 input tied low in the functional clock tree. Specifying “with_clock_gating_cell” will explicitly request the clock gating library cell to gate the bist clock.

- min_misr_segment_bits : 24 | 256 ;

A property that specifies the number of bits contained within each segment that composes the MISR. When the default setting of 24 is specified, the MISR will be composed of 24-bit and 32-bit segments. The tool will select the combination of segments, based on ROM word size, to minimize the total length of the MISR. A setting of 256 will only utilize 256-bit segments for the MISR and reduce aliasing. The 256 setting is only supported for programmable controllers.

AlgorithmResourceOptions

Specifies properties related to memory BIST controller hardware to support custom algorithm implementation.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        Controller(id) {
            AlgorithmResourceOptions { // *DefSpec
                soft_instruction_count : int ; // default: 0
                data_register_bits : int | auto ;
                counter_a_bits : int | auto ;
                delay_counter_bits : int | auto ;
                preserve_microcode_initial_values : on | off ;
                soft_algorithm_address_min_max : on | off ;
                max_data_inversion_address_bit_index : int | max_index ;
                                                // default: 0
                a_equals_b_command_allowed : on | off ;
                address_segment_x0_y0_allowed : on | off ;
                max_x0_segment_bits : 1 | auto ;
                max_y0_segment_bits : 1 | auto ;
            }
        }
    }
}
```

Description

A wrapper that specifies the amount of hardware to include into the memory BIST controller to support custom algorithm implementation. The more you add, the more flexible your algorithms can be but the more hardware it takes to implement the controller.

This entire wrapper can be specified in the [DefaultsSpecification/DftSpecification/MemoryBist](#) wrapper such that it gets created as you want when you run the [create_dft_specification](#) command.

Arguments

- `soft_instruction_count : int;`

Specifies the number of instructions you want available for programming your soft programmable algorithms. The default value is 0 which means that only algorithms hardened into the controller will be usable.

The `soft_instruction_count` property specifies the number of instruction words that will be implemented in the microprogram memory of the soft programmable controller. The number of bits per instruction word depends on many factors such as the logical address map, number of operations, and the number of instructions.

The soft programmable controller supports the downloading of programmable algorithms at run time. You can load any soft algorithm at run time using the `apply_algorithm` property inside the `MemoryBist/Controller/AdvancedOptions` wrapper of the [PatternsSpecification](#).

The number of instructions in the soft algorithm must not exceed the value specified for this property.

Specifying the soft_instruction_count property is a trade-off between the amount of instructions your custom soft programmed algorithm can have and area. If a large number of instructions is specified, there is a high cost in hardware to implement the microprogram memory. However, if the number of instructions is small, the types of algorithms that can be implemented is significantly reduced.

- `data_register_bits : int | auto ;`

The data_register_bits property allows you to specify the number of bits in the write and expect data registers. The number of bits in the write and expect data registers can represent the entire width of the memory data bus or a segment of the memory data bus. If the data register size is wider than the memory data bus width, the data pattern will be truncated to fit the bus width. If the data register size is narrower than the memory data bus width, the data pattern will be replicated to fill the bus width. The default value of auto means 2 bits if the hard-coded algorithms are limited to SMarch, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, and SMarchCHKBvcd. If other algorithms are built into the controller, the default value is the number of bits on the widest memory tested by the controller.

- `counter_a_bits : int | auto ;`

The optional counter_a_bits property specifies the width of the general purpose CounterA counter. The maximum width that can be specified is 64 bits. The default value of auto gets converted to an integer that is the largest of the following:

Number of counter bits required to count the number of rotate commands to move a bit across the length of the data register.

Number of counter bits required to count the number of rotate commands to move a bit across the length of the Z address register plus the Y address register plus the X address register.

- `delay_counter_bits : int | auto ;`

The optional delay_counter_bits property specifies the number of bits in the general purpose DelayCounter. The default auto value for delay_counter_bits gets converted into an integer based on the MemoryType property of the Memory library wrapper—see “[Tessent MemoryBIST Library Algorithms](#)” in the *Tessent MemoryBIST User's Manual*. If the controller only test memories of type SRAM or ROM, the auto value gets converted to 0. If the controller tests one or more memories of type DRAM, the auto value gets converted to a number large enough to meet the refresh requirements of the DRAM memories taking into account the period of the clock used by the controller.

- `preserve_microcode_initial_values : on | off ;`

When the property is specified to on and the memory BIST controller is soft programmable (the property soft_instruction_count is greater than 0) and at least one of the memories tested by the controller is a multi-port memory other than a 1R1W configuration, the

`preserve_microcode_initial_values` property inserts shadow registers for the following controller registers:

X, Y, and Z segments of Address Register A

X, Y, and Z segments of Address Register B

Write data register

Expect data register

When a soft-coded algorithm is applied at runtime, the microcode values are loaded once into the soft programmable controller at the start of the test. The shadow registers replicate the initial values of the registers above. For multi-port memory testing, the algorithm is applied for each test port. Since the soft-coded algorithm is not reloaded for each test port, the microcode initial values must be preserved as the actual registers might be manipulated during the algorithm execution.

Depending on the memory word width and the number of words, the shadow registers might add significant area overhead to the soft programmable controller. An alternative approach is to specify this property to off and use the `freeze_test_port` property inside the [AdvancedOptions](#) wrapper to apply the soft-coded algorithm to one specific test port of the multiport memory at a time.

- `soft_algorithm_address_min_max : on | off ;`

The `soft_algorithm_address_min_max` property allows the user to specify a user defined address count range in the soft algorithms. This must be set to on if any of the following properties will be specified in the soft algorithm:

- `load_bank_address_min`
- `load_bank_address_max`
- `load_column_address_min`
- `load_column_address_max`
- `load_row_address_min`
- `load_row_address_max`

- `max_data_inversion_address_bit_index : int | max_index ;`

The `max_data_inversion_address_bit_index` property determines the size of the multiplexer used by the `invert_data_with_row_bit` and `invert_data_with_column_bit` properties inside the `DataGenerator` wrapper of custom Algorithm wrapper—see “[Defining a Custom Algorithm](#)” in the *Tessent MemoryBIST User’s Manual*. The address bit selection logic inside the memory BIST controller creates critical paths that may affect timing closure when synthesizing at high clock frequencies. You should only specify a non-zero value for this property unless you intend to use the `invert_data_with_row_bit` or

invert_data_with_row_bit property with values other than r[0] and c[0] respectively. When unspecified, this property defaults to 0. Valid values are as follows:

integer — limits the highest bit index of the row and column address that can be specified for the invert_data_with_row_bit and invert_data_with_column_bit properties. Valid values are from 0 to N.

max_index — allows any row or column address bits to be selected for write or expect data register inversion.

- a_equals_b_command_allowed : on | off ;

The a_equals_b_command_allowed property specifies whether the comparator that is used for the address_a_equals_b property (used in the InstructionData wrapper) is generated inside the memory BIST controller—see “[Instruction/AdvancedOptions](#)” in the *Tessent MemoryBIST User’s Manual*. The presence of the address comparator in the memory BIST controller creates critical paths that may affect timing closure when synthesizing at high clock frequencies. It is recommended to set the pipeline_controller_outputs property (see [AdvancedOptions](#)) property to on when this property is set to on. If you do not intend to define custom algorithms that use the address_a_equals_b property, the property should be set to off to facilitate timing closure.

When a_equals_b_command_allowed is set to off, custom algorithms cannot use the address_a_equals_b feature. That is, the address_a_equals_b property must be off in all InstructionData wrappers—see “[Instruction/DataCommands](#)” in the *Tessent MemoryBIST User’s Manual*.

- address_segment_x0_y0_allowed : on | off ;

The address_segment_x0_y0_allowed property allows the row and column address counters to be further segmented. The row address counter may be sub-divided into a X1 address counter and a X0 address counter. The column address counter may be sub-divided into a Y1 address counter and a Y0 address counter. The algorithm can use these address segments to achieve complex address count sequences. The size of the individual address segment is configured through the properties max_x0_segment_bits and max_y0_segment_bits in the algorithm.

- max_x0_segment_bits : 1 | auto ;

The optional max_x0_segment_bits property specifies the limit of the X0 address segment size.

When max_x0_segment_bits is set to 1, the address counter logic is modified so that all critical single-cycle paths to the other registers of the memory BIST controller are removed. Corresponding multi-cycle constraints are added to relax these paths during synthesis, timing analysis, and layout. This facilitates the timing closure of the controller hardware when operating at high clock frequency and when incrementing/decrementing the address in consecutive clock cycles of custom algorithms. The value of 1 for max_x0_segment_bits is suitable for all Mentor Graphics library algorithms.

Setting max_x0_segment_bits to Auto allows the implementation of address sequences incrementing or decrementing the address by 2 or more in custom algorithms. The address

can be incremented or decremented in consecutive cycles of custom algorithms but the presence of additional critical single-cycle paths might limit the maximum operational speed of the controller, especially for soft-programmable controllers which allow all possible X0 address segment sizes. If a hard-programmable controller is implemented, the tool automatically determines the maximum size of the X0 address segment required. This allows to minimize area and the number of critical single-cycle paths.

Please refer to the description of `column_address_count_enable` and `row_address_count_enable` in the [Cycle/AdvancedSignals](#) wrapper properties for additional usage conditions concerning the implementation of custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

The following usage conditions apply:

- Refer to the `row_address_count_enable` usage conditions for the `max_x0_segment_bits` property usage when implementing custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.
- `max_y0_segment_bits : 1 | auto ;`

The optional `max_y0_segment_bits` property specifies the limit of the Y0 address segment size.

When `max_y0_segment_bits` is set to 1, the address counter logic is modified so that all critical single-cycle paths to the other registers of the memory BIST controller are removed. Corresponding multi-cycle constraints are added to relax these paths during synthesis, timing analysis, and layout. This facilitates the timing closure of the controller hardware when operating at high clock frequency and when incrementing/decrementing the address in consecutive clock cycles of custom algorithms. The value of 1 for `max_y0_segment_bits` is suitable for all Mentor Graphics library algorithms.

Setting `max_y0_segment_bits` to Auto allows the implementation of address sequences incrementing or decrementing the address by 2 or more in custom algorithms. The address can be incremented or decremented in consecutive cycles of custom algorithms but the presence of additional critical single-cycle paths might limit the maximum operational speed of the controller, especially for soft-programmable controllers which allow all possible Y0 address segment sizes. If a hard-programmable controller is implemented, the tool automatically determines the maximum size of the Y0 address segment required. This allows to minimize area and the number of critical single-cycle paths.

Please refer to the description of `column_address_count_enable` and `row_address_count_enable` in the [Cycle/AdvancedSignals](#) wrapper properties for additional usage conditions concerning the implementation of custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

The following usage conditions apply:

- Refer to the `column_address_count_enable` usage conditions for the `max_y0_segment_bits` property usage when implementing custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

MemoryCluster

Specifies the controller properties to generate a memory BIST controller to test memories that use a shared bus interface.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        Controller(id) {
            MemoryCluster(cluster_id) {
                instance_name : instance_name ;
                memory_cluster_library_name : library_name ;
                pipeline_cluster_inputs : on | off ; // *DefSpec
                pipeline_cluster_outputs : on | off ; // *DefSpec
                memory_access_level : auto | logical | physical ;
                                         // *DefSpec
                repair_analysis_present : auto | off ; // *DefSpec
                max_repair_group_size : unlimited |
                                         {int[kilobits | megabits]} ;
                                         // *DefSpec
                repair_group_scope : logical_memory |
                                     physical_memory | controller ;
                                         // *DefSpec
                repair_sharing : on | off ;
            }
        }
    }
}
```

Description

A wrapper that specifies the controller properties to generate a memory BIST controller to test memories that use a shared bus interface. The [MemoryCluster](#) and [Step](#) wrappers are mutually exclusive. The shared bus interface provides common access to multiple memories. Please refer to the [Tessent MemoryBIST User's Manual](#) for further details.

Arguments

- **instance_name** : *instance_name* ;
A property that specifies the instance path of a memory cluster that will be connected to a shared bus memory BIST controller
- **memory_cluster_library_name** : *library_name* ;
An optional property that specifies the cluster library template name that will be used to generate a shared bus memory BIST controller. This is only needed if you wish to generate a memory BIST controller without insertion.
- **pipeline_cluster_inputs** : on | off ;
A property that specifies whether the memory cluster inputs from the memory BIST controller will be pipelined with a single pipelining stage to help with meeting timing closure.

- pipeline_cluster_outputs : on | off ;

A property that specifies whether the memory cluster outputs to the memory BIST controller will be pipelined with a single pipelining stage to help with meeting timing closure.

- memory_access_level : auto | logical | physical ;

A property that specifies the view of all memories in the memory cluster that the memory BIST controller operates on. The memories accessed through the shared bus interface are called logical memories. A logical memory is an address space composed of one or more physical memories.

When the value is set to logical, the physical composition of the logical memory is irrelevant. All memories that form the logical memory are tested in one pass of the test algorithm. Repair and diagnosis information will be reported with respect to the address space and I/O of the logical memory. You use this option if each logical memory contains one physical memory. It is also suitable if you want to implement repair sharing for the physical memories that you integrate into the logical memory.

When the value is set to physical, each physical memory composing a logical memory is tested independently of each other. This option provides repair and diagnosis resolution mapped to the physical memories but it increases the area overhead.

When the value is set to auto, the access level is determined for each logical memory from the specification of its logical library. The auto value is converted to physical if all of the following conditions are met: (a) the logical memory contains one or more physical memories—at least one PhysicalToLogicalMapping wrapper is present, (b) the logical memory defines no repair—the RedundancyAnalysis wrapper is not present, and (c) the logical memory defines no address segmentation —the AddressCounter wrapper is not present.

- repair_analysis_present : auto | off ;

A property that specifies if repair analysis logic is generated for the indicated Shared Bus memory cluster. If this property is set to auto and the Shared Bus logical or physical memory's library descriptions (as specified by memory_access_level) contain at least one RedundancyAnalysis/SpareElement wrapper, the repair analysis logic is generated. Refer to the “[RedundancyAnalysis](#)” section in the *Tessent MemoryBIST User's Manual* manual for more information.

- max_repair_group_size : unlimited | {int[kilobits | megabits]} ;

A property that specifies the total memory size limit on a shared bus memory group when repair sharing is enabled with [set_memory_instance_options](#). This property is used during logical/physical memory expansion when the [process_dft_specification](#) command is run. The default value of “unlimited” indicates that any number of compatible shared bus memories can be assigned to repair groups. The int[units] pair specifies the maximum size allowed for a repair group. Any shared bus memory that would cause this limit to be exceeded will be assigned to a different repair group.

- repair_group_scope : logical_memory | physical_memory | controller ;

A property used to specify how repair grouping is conducted during logical/physical memory expansion for shared bus memories when the [process_dft_specification](#) command is run and repair sharing is enabled with [set_memory_instance_options](#).

The selected value impacts the repair grouping as follows:

- logical_memory — Specifies the repair groups are restricted to compatible physical memories **within** a logical memory without exceeding the size specified by the max_repair_group_size property in the MemoryCluster wrapper.
- physical_memory — Specifies the repair group is restricted within a physical memory. Memories with multiple [RowSegment](#) and/or [ColumnSegment](#) wrappers will be considered as having a single RowSegment and/or ColumnSegment wrapper for redundancy analysis as long as the memory size does not exceed the size specified by the max_repair_group_size property in the MemoryCluster wrapper.
- controller — Specifies the repair group is restricted to all memories tested by the controller. The repair group cannot contain memories tested by other controllers. Memories are automatically assigned to repair groups as long as the memory size does not exceed the size specified by the max_repair_group_size property in the MemoryCluster wrapper.

- repair_sharing : on | off ;

A property that will globally enable and disable repair sharing for shared bus memories that have “-repair_sharing auto” specified, which is the default. The “-repair_sharing” value can be changed from “auto” for a memory instance with the [set_memory_instance_options](#) command.

RepairOptions

Provides repair-related options and properties.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        Controller(id) {
            RepairOptions {
                fuse_set_extraction_sequence : address_before_fuse_map | fuse_map_before_address ;
                row_bira_location : controller | follow_comparators ;
            }
        }
    }
}
```

Description

The RepairOptions wrapper specifies what to shift out and monitor with repairable memories, as well as the spare allocation method to use.

Arguments

- **fuse_set_extraction_sequence** : address_before_fuse_map | fuse_map_before_address ;
A property that defines the setup chain sequence of the Column/IO BIRA registers that report the faulty column address and the IO decode value. This property specifies whether the faulty column address or the IO decode value will be shifted out first when the repair analysis information is extracted from the memory BIST controller. Valid values are as follows:
 - AddressBeforeFuseMap — the faulty column address will be shifted out before the IO decode value. The MSB of the IO decode register connects to the LSB of the faulty column address register.
 - FuseMapBeforeAddress — the IO decode value will be shifted out before the faulty column address. The MSB of the faulty column address register connects to the LSB of the IO decode register.
- **row_bira_location** : controller | follow_comparators ;
A property that specifies where the Row BIRA module will be located. The Row BIRA module can be located in the same module as the comparators or it can be forced to be inside the memory BIST controller.
When shared comparators are used (comparator_location:shared_in_controller), the Row BIRA module is always located inside the memory BIST controller. However, when local comparators are used (comparator_location:per_interface), the Row BIRA module can be placed inside the interface or inside the controller.

For a memory BIST controller with several memories with Row BIRA, placing the Row BIRA module inside the controller results in a reduction of area due to the sharing of common redundancy analysis logic and pipelining stages.

DiagnosisOptions

Specifies the diagnosis options that affect diagnostic hardware generation for the memory BIST controller.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        Controller(id) {
            DiagnosisOptions { *DefSpec
                comparator_selection_mux : on | off | auto ;
                go_status : auto | per_memory ;
                StopOnErrorOptions {
                    failure_limit : int | off ;
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the diagnosis options that affect diagnostic hardware generation for the memoryBIST controller.

Arguments

- **comparator_selection_mux** : on | off | auto ;

A property that specifies whether bit-level diagnostic resolution is maintained when local_comparators_per_go_id or shared_comparators_per_go_id is set to a value greater than 1. The valid values are:

on — specifies that a multiplexer is inserted at the input of each go_id register to select the output of only one comparator at a time and provide bit-level diagnostic resolution when the local_comparators_per_go_id or shared_comparators_per_go_id property is set to a value greater than 1.

off — specifies that bit-level diagnostic resolution is not required.

auto — infers “on” when the StopOnErrorOptions/failure_limit property is not set to “off”; otherwise, infers “off”.

- **go_status** : auto | per_memory ;

A property that specifies how a pass/fail status per individual memory is generated for the current controller.

The per-memory status register allows you to determine which memories failed without having to examine the individual compare status registers (GO_ID bits). The valid values are:

auto — a per-memory pass/fail status register is generated when memories with redundancy are present for the current MemoryBIST controller. If there are no

memories with redundancy for the current controller, then the per-memory status register is not added. This is the default value.

per_memory — A per-memory pass/fail status register is added for all memories on the current MemoryBIST controller, even if there are no memories with redundancy.

- **StopOnErrorOptions/failure_limit:** *int* | off ;

A property that specifies the maximum number of failures that the memory BIST controller can log in the Stop-On-Nth-Error mode of diagnosis. The default value becomes 4096 when the controller tests at least one memory of type SRAM or DRAM. It becomes “off” when the controller tests only memories of type ROM. The error counter is no longer used for ROM diagnosis as it was in the LV flow. Instead the functional debug mode is used to perform ROM diagnosis.

Step

Specifies the properties for memory BIST steps.

Usage

```
DftSpecification(module_name,id) {
    MemoryBist {
        Controller(id) {
            Step { // repeatable
                algorithm : algo_name ;           // *DefSpec
                operation_set
                comparator_location
                bist_data_in_pipelining
                bist_data_out_pipelining
                MemoryInterface(id) {
                    instance_name
                    memory_library_name
                    repair_analysis_present
                    repair_group_name
                    scan_bypass_logic
                    local_comparators_per_go_id
                    rom_content_file
                    output_enable_control
                    observation_xor_size
                }
                ReusedMemoryInterface(id) {
                    instance_name
                    reused_interface_id
                    repair_group_name
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the properties for memory BIST steps. These properties are used to generate specific controllers' configurations. The [Step](#) and [MemoryCluster](#) wrappers are mutually exclusive. See also “[Tessent Core Description](#)” on page 3755 for Memory wrapper details.

Arguments

- algorithm : algo_name | [from_library](#) ;

The ram_algorithm property specifies the name of the algorithm that is applied to all memories tested in this step. If the “from_library” value is used, the algorithm specified in the Memory library description of all memories tested by the controller in the given Step is used. An error is generated if the “from_library” value is used and the memories tested by the controller in the given Step do not have the same value for the algorithm property.

- `operation_set : opset_name ;`

The `operation_set` property specifies the name of the operation set that is applied to all memories tested in this step. If the “from_library” value is used, the OperationSet specified in the Memory library description of all memories tested by the controller in the given Step is used. An error is generated if the “from_library” value is used and the memories tested by the controller in the given Step do not have the same value for the `OperationSet` property.

- `comparator_location : shared_in_controller | per_interface ;`

The `comparator_location` property specifies where comparators for the memory BIST controller are placed, with the default being the placement of the comparators in the memory interface. Using the `per_interface` value greatly reduces the routing area needed to bring the BIST data signals from the memory interfaces to the controller. You also get per memory diagnostic resolution in go-nogo tests even when the controller has more than one step when the comparators are inside the interfaces. You can use the `local_comparators_per_goid` property to reduce the area overhead.

- `bist_data_in_pipelining : on | off | int;`

The `bist_data_in_pipelining` property defines the number of pipeline stages to be added to all BIST inputs in the memory interface. Setting the value “on” is equivalent to 1 pipeline stage. You can use this property to facilitate timing closure on paths from the controller. The dedicated pipeline flip-flops allow the layout tool to place the flip-flops at the optimal distance for each memory. Typically, a single pipeline stage is sufficient. The `MemoryBist/Controller/AdvancedOptions/pipeline_controller_outputs` option is much less costly and is typically good enough to meet timing. The pipeline registers are added in the memory interface on the BIST signals from the controller. Some signals propagate to the memory while other signals are internal to the memory interface or controller and control the test execution. Examples of test signals that are pipelined when `bist_data_in_pipelining` is specified are address and control to the memory, write data pattern, expect data pattern, compare enable, and checkerboard pattern enable.

When using the `bist_data_in_pipelining` property, you do not need to update any operation set because the pipelining is taken into account when writing to and reading from the memory.

- `bist_data_out_pipelining : on | off | per_port;`

A property that specifies whether the BIST data from the memory to the comparators will be pipelined with a single pipelining stage. The pipelining stage is inserted in the memory interface. The valid values and their effects are as follows:

`on` — pipelines the BIST data out to the comparators. You specify `on` if the BIST data output does not have time to propagate to the comparator. Pipelining might be required when the comparators are located in the memory BIST controller due to the physical distance between the memory controller and memory interface.

`off` — does not pipeline the BIST data out.

`per_port` — pipelines the BIST data out per memory output port. For multi-port memories, the data from each output port is multiplexed into a single BIST data out to

the comparators. You specify `per_port` to insert separate pipelining flip-flops on each data output port before the multiplexer.

- `MemoryInterface(id)/instance_name : inst_name ;`

A property that specifies the instance name of the memory interface associated with the given memory. This property is required when running the [process_dft_specification](#) command without the `-no_insertion` option. In this case, the value of the `tcd_memory_name Instance` attribute attached to the instance defines the Memory library associated to that instance. When you use the `-no_insertion` flow, the elaborated design that contain the instantiated memories is optional. When the design is not present, the Memory library associated with the `MemoryInterface` is defined using the `memory_library_name`.

- `MemoryInterface(id)/memory_library_name : mem_lib_name ;`

A property that specifies the name of the memory library associated with the memory tested by the memory interface. This property is only used and needed if you are running the [process_dft_specification](#) command with the `-no_insertion` option and the design is not elaborated in memory. Otherwise, the `memory_library_name` is extracted from the `tcd_memory_name` attribute attached to the instance referenced by the `MemoryInterface(id)/instance_name` property.

- `MemoryInterface(id)/repair_analysis_present : auto | off ;`

A property that specifies whether repair analysis logic is generated for the memory tested by the memory interface. If set to `auto` and the memory library description associated to the memory tested by the memory interface contains at least one `RedundancyAnalysis/SpareElement` wrapper, the repair analysis logic is generated. See “[Tessent Core Description](#)” on page 3755 for `RedundancyAnalysis` wrapper information.

- `MemoryInterface(id)/repair_group_name : none | group_name ;`

A property that specifies the group name for the memory interfaces that will share a `BIRA/BISR` module. This property is automatically filled in when the [create_dft_specification](#) command is run and repair sharing is enabled with [set_memory_instance_options](#). The user can update this property to manually change the repair group assignments.

- `MemoryInterface(id)/scan_bypass_logic : async_mux | none | sync_mux | from_library ;`

A property that specifies when and how memories are bypassed during scan testing. Bypassing memories enables testing of the user interface logic to and from these embedded memories as well as to and from the memory `BIST` controller and memory interface circuitry.

If the specified value for `MemoryInterface(id)/scan_bypass_logic` equals “`sync_mux`”, “`async_mux`” or “`none`”, it overrides the `TransparentMode` property in the Memory library description. If the value is not specified or set to “`from_library`”, the value in the Memory library file takes effect.

Valid values are as follows:

`async_mux` — inserts an additional multiplexer on the memory interface `DataOut` ports so that data is directly transferred from the `DataIn` ports to the `DataOut` ports. This

setting enables combinational ATPG tools to test the interface between the user logic and the memory. It is not recommended for at-speed testing because the timing paths are not consistent with the functional mode.

sync_mux — inserts an additional multiplexer and flip-flop on the interface DataOut ports that enable ATPG tools to test the interface between the user logic and the memory without having to enable testing through the memory. This mode is very efficient for ATPG and provides very consistent timing paths between test and functional modes.

none — specifies that no bypass logic will be added in the memory interface. Specify None if you have a bidirectional data bus or the memory implements internal bypass logic.

from_library — specifies that the bypass logic will be generated according to the TransparentMode property setting of the Memory library file.

- **MemoryInterface(id)/local_comparators_per_go_id : int | all ;**

A property that specifies the number of comparators to share with a given go_id register located in the interface. You can use this feature to manage the area overhead by reducing the number of registers used to log the comparator status. However, using a value other than 1 complicates and lengthens diagnosis with bit-level resolution as the test needs to be repeated as many times as there are comparators sharing a common go_id register.

The value should be set to 1 to enable reading complete memory words in a single read operation. Also, the MemoryBist/Controller/[AlgorithmResourceOptions](#)/data_register_bits property should be set to an appropriate value based on the complexity of the patterns to be written to memory. For complete control of the write data, this property should be set to the number of bits of the widest memory tested by the controller.

The default value is 1. Specifying the value “all” infers that there will be one go_id register for all comparators associated with a memory.

- **MemoryInterface(id)/rom_content_file : file_path ;**

A property that specifies the location of the ROM content file. The path is absolute or relative to the working directory. This property is mandatory when the memory tested by the memory interface is of type ROM. The ROM content file is used to compute the signatures associated with all algorithms applicable to ROMs and built into the controller. See the [Step](#)/Algorithm property and the [AdvancedOptions](#)/extra_algorithms to understand which algorithm gets built into the controller.

- **MemoryInterface(id)/output_enable_control : always_on | system ;**

The output_enable_control property should only be specified when the design is not elaborated. Otherwise, it is recommended to let Tesson Shell analyze the design and infer the proper output_enable_control setting based on design connectivity. When the design contains many non-unique instances of the same memory interface with different output enable connection configurations, Tesson Shell will apply the value that accommodates all memory instances.

The `output_enable_control` property is used to determine the type of data output stage (2-state or 3-state) when generating the memory interface modules, as well as the logic that controls the output enable ports of the memory and interface modules. When set to `always_on`, a 2-state output multiplexer is generated. When set to `system`, a 3-state multiplexer is generated.

Specifying The `output_enable_control` Based on Design:

Set `output_enable_control` to `always_on` when the memory output enable port is tied to a constant value in the functional design. This maintains the memory data output drivers (tri-state) into an always on state. The memory BIST interface generated for this memory intercepts the data output with a standard 2-state mux and drives the values from the bypass registers during scan if the memory bypass is selected during scan. Else, the interface will select the memory output during functional mode or when applying multi-load scan patterns to memory. This will not cause any net contention with other memories in the design, and is the most common situation.

Set `output_enable_control` to `system` when the memory output enable port is driven by functional logic. This is typically the case when many memories are connected to the same data bus. It is assumed that the functional logic enables a single memory output at any time and avoids bus contentions. The memory BIST interface generated for this memory must also have a 3-state output. During scan mode, the memory interface observes the memory output enable signal and either enables the output drivers from the memory interface (when memory bypass is enabled), or the memory output drivers (when applying multi-load scan patterns to memory).

- `MemoryInterface(id)/observation_xor_size : auto | off ;`

A property that specifies whether or not to insert scan observation points for the address and control ports in the memory interface. The `auto` setting combines the signals based on the `observation_xor_size` property in the `MemoryBist/Controller/AdvancedOptions` wrapper. The `off` setting disables the observation logic insertion. The default setting is `auto`.

- `ReusedMemoryInterface(id)/instance_name : inst_name ;`

A property that is used to associate a memory interface module with another memory instance. You can use this wrapper when the memory instance has the same module name as the one for which the reused memory interface is for. The `create_dft_specification` command will use this wrapper when the parent module of the memory instance is not unique to avoid having to uniquify the parent module during insertion. It will also use this wrapper when the memory instance is inside a generate loop to avoid having to unroll the loop during insertion. This property is optional when the `process_dft_specification` command is run with the `-no_insertion` option.

- `ReusedMemoryInterface(id)/reused_interface_id : [ctrl_id:]mem_interface_id ;`

A property that specifies the memory interface that is to be reused by the given memory instance. The value is the id of the controller wrapper, a colon, and the id of the `memory_interface` wrapper that is reused. The id of the controller and the colon is optional when the reused `MemoryInterface` belongs to the same controller as the `ReusedMemoryInterface` wrapper.

- ReusedMemoryInterface(id)/repair_group_name : none | *group_name* ;

A property that specifies the group name for the reused memory interfaces that will share a BIRA/BISR module. This property is automatically filled in when the [create_dft_specification](#) command is run and repair sharing is enabled with [set_memory_instance_options](#). The user can update this property to manually change the repair group assignments.

ReusedController

Specifies the insertion of the second instance of a memory BIST controller.

Usage

```
DftSpecification(module_name,id) {
    MemoryBist {
        ReusedController(id) { // repeatable
            parent_instance : inst_path ;
            leaf_instance_name : inst_name ;
            clock_domain_label : name ;
            reused_controller_id : id ;
            Step { //repeatable
                MemoryInterface(id) { // repeatable
                    instance_name : name ;
                }
            }
            MemoryCluster(cluster_id) {
                instance_name : name ;
            }
        }
    }
}
```

Description

A wrapper that specifies the insertion of the second instance of a memory BIST controller. The [create_dft_specification](#) command uses this wrapper when it detects that two controllers test the exact same set of memory modules and that the common ancestor of each of those two sets are repeated instances of a single module. To avoid having to lift the controller into a unique location or having to uniquify the repeated module, the memory BIST controller module is reused. The ReusedController/Step and ReusedController/MemoryCluster wrappers are mutually exclusive.

Arguments

- **parent_instance : *inst_path* ;**
An optional property that specifies where the memory BIST controller is instantiated in the design. When unspecified, it defaults to a null string which is interpreted as the common instance ancestor of all memories tested by the controller. See the description of the [get_common_parent_instance](#) command to see how the common ancestor is computed.
- **leaf_instance_name : *inst_name* ;**
An optional property that specifies the memory BIST controller instance name explicitly. By default, an instance name is automatically generated by appending the “_inst” string to the module name. This property must be identical in all Controller/ReusedController wrapper which have their parent_instance specified or inferred to point to instances of the same repeated module.

- `clock_domain_label : name ;`
A property that specifies the label (see [add_clocks](#)) of the clock domain used by the memory BIST controller and all the associated memories. Any memory tested by the controller which has a clock port sourced by a different clock domain will have a multiplexer added in its interface module to inject the clock used by the controller when running memory BIST.
- `reused_controller_id : id ;`
A property that specifies the id of a controller wrapper that is to be reused.
- `Step/MemoryInterface(id)/instance_name : name ;`
A property that specifies the instance name of the memory to be tested by the given memory interface in the given [Step](#).
- `MemoryCluster(cluster_id)/instance_name : name ;`
A property that specifies the instance name of the memory cluster to be tested by the given controller.

BistAccessPort

A wrapper that specifies where the memory BIST BIST Access Port (memory BIST BAP) module is instantiated in the design.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        BistAccessPort {
            parent_instance      : inst_path ;
            leaf_instance_name   : inst_name ;
        }
    }
}
```

Description

The BistAccessPort wrapper provides the capability to locate the memory BIST BAP instance at a specified location in the design hierarchy. The memory BIST BAP provides control signals to memory BIST controllers, collects their GO and DONE status signals and provides access to their internal registers. A memory BIST BAP is inserted for each pass when doing multi-pass insertion.

Arguments

- **parent_instance** : *inst_path*

An optional property that specifies the hierarchical path where the memory BIST BAP module is instantiated in the design.

When unspecified, it defaults to a null string that is interpreted as the common instance ancestor of all memory controllers driven by the memory BIST BAP. See the description of the [get_common_parent_instance](#) command to see how the common ancestor is computed. When *inst_path* is specified as “.”, the memory BIST BAP is instantiated in the top module of the design.

This property is not used or needed when the [process_dft_specification](#) command is run with the -no_insertion option.

- **leaf_instance_name** : *inst_name*

An optional property that specifies the memory BIST BAP instance name explicitly. By default, an instance name is automatically generated by appending the “_inst” string to the memory BIST BAP module name, which itself defaults to the following:

design_name_design_id_tessent_mbist_bap

The *design_id* is specified with the [set_context](#) -design_identifier command for the current design.

This property is not used or needed when the [process_dft_specification](#) command is run with the -no_insertion option.

TCKInjectionPoints

Specifies the locations in the design where a mux will be inserted to inject TCK onto the functional clock trees for TCK mode implementation.

Usage

```
DftSpecification(module_name, id) {
    MemoryBist {
        TCKInjectionPoints {
            ClockMux (mux_id) {
                leaf_instance_name : instance_name ;
                node : node_name ;
            }
        }
    }
}
```

Description

The TCKInjectionPoints wrapper defines the locations where muxes will be inserted to inject TCK. The TCK that will be injected is the one used by the specified [MemoryBist/ijtag_host_node](#). These muxes will be controlled by the DFT signal “tck_select”, which is asserted as needed during patterns steps by setting the [Patterns tck_clock_only](#) property to “on”.

Arguments

- `leaf_instance_name : instance_name ;`

A string property that specifies the leaf instance name of the mux. A prefix is added to the specified *instance_name* as follows:

`<persistent_clock_cell_prefix>_tck_mux_instance_name`

If this property is not specified, it defaults to:

`<persistent_clock_cell_prefix>_tck_mux_<design_name>_<dft_spec_id>_<mux_id>_inst`

The “*dft_spec_id*” string is the “*id*” specified in the DftSpecification wrapper, and the “*mux_id*” is the string specified in the ClockMux wrapper. Tessent Shell checks for name conflicts before instantiating the mux and, if needed, applies the unification suffix specified by the [set_insertion_options](#) command.

- `node : node_name ;`

A required property that specifies the location of a port, pin or net name to be intercepted with a mux instance.

OCC

The OCC DftSpecification syntax defines the configuration of the OCC to be generated and inserted into the design.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        ijtag_host_interface      : host_name | none;
        capture_trigger            : auto | shift_en | capture_en;
        static_clock_control       : auto | off | [internal | external | both];
        capture_window_size        : int ; // default: 3
        type                      : standard | parent | child;
        internal_clock_gater       : on | off | auto;
        shift_only_mode            : on | off | auto;
        kill_clock_mode             : on | off | auto;
        include_clocks_in_icl_model : on | off ;
        leaf_instance_name          : instance_name ;
        Interface {
        }
        Connections {
        }
        Controller(id) {
        }
    }
}
```

Description

The OCC DftSpecification syntax defines the configuration of the OCC to be generated and inserted into the design.

In modern designs, OCC circuits are commonly used to manage clocks during test. Such clock controllers can generate slow-speed or at-speed clock sequences under the control of ATPG process. Tesson OCC is an implementation of a clock controller created by Tesson Shell that has been designed to meet the requirements of scan test for ATPG, Logic BIST, EDT, and Low Pin Count Test.

The OCC follows the `use_rtl_synchronizer_cell` property to decide if a real or an RTL cell synchronizer cell is used. The OCC also follows the `use_rtl_synchronizer_cell` property to use a synchronizer cell with or without a reset port. The use of the asynchronous reset removes the need to add extra shift cycles in the parallel load simulations even when you have capture clocks which are slower than the shift clock.

Note

 The Sync cell is instantiated with the following leaf instance prefix:

*persistent_cell_prefix*test_ntc_

where *persistent_cell_prefix* is specified using the [persistent_cell_prefix](#) property of the DftSpecification wrapper. You can use this special prefix to locate them in the design if you want to disable their notification during logic test Verilog simulations.

Refer to “[On-Chip Clock Controller Design Description](#)” and the “[OCC schematic](#)” in the *Tessent Scan and ATPG User’s Manual* for more information.

Note

 The standard OCC is the recommended OCC for use in hierarchical cores. For more information, see “[Core OCC Recommendation](#)” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- *ijtag_host_interface* : *host_name* ;

A property that specifies the host scan interface to which the clock controllers will be connected. The default value is “none” which means that the static controls are primary inputs on the control. With the “none” value, you can use the Connections/
[StaticExternalControls](#) wrapper to specify where to connect them. If you are using IJTAG with all its advantages with respect to the test_setup automation, use the [create_dft_specification -sti_sib_list](#) switch with a value of “Occ” to have the DftSpecification created with an Sib ready to host the OCC controllers. You then point to it using “Sib(Occ)” as the *ijtag_host_interface* value.

- *capture_trigger* : *enum* ;

A property that specifies whether *shift_en* or *capture_en* is used as the trigger to generate programmable capture pulses. The *capture_en* version is used when some internal test logic generates a separate *capture_en* signal, such as the LPCT or LBIST controller.

The default is auto.

Choose one from the following:

auto — Automatically determines whether *shift_en* or *capture_en* is used as the trigger. When the OCC wrapper is used with a LogicBist or an LpcfType3 wrapper in the same insertion pass (that is, the wrappers are present in the same DftSpecification) auto resolves to the *capture_en* value; otherwise it resolves to *shift_en*.

capture_en — The OCC controllers will infer the *capture_en* port.

shift_en — The OCC controllers will infer the *shift_en* port.

- static_clock_control : *enum* ;

A property that specifies whether capture clock generation is available outside of ATPG control.

There is a dependency between this property and the [StaticExternalControls](#) wrapper's static_clock_control_mode property. By default the OCC wrapper's static_clock_control property defaults to off; you must set this property to a value other than off in order to generate the static clock control pins on the OCC module.

Choose one from the following:

auto — When both OCC and LogicBIST are included in the same insertion pass (that is, the wrappers are present in the same DftSpecification), auto resolves to external when LogicBIST is present; otherwise, it resolves to off.

off — Capture clock generation is only under ATPG control, specifically static programmability. This is the default.

internal — Non-ATPG control capture clock generation is available through an IJTAG TDR in the OCC (static programmability).

external — Non-ATPG control capture clock generation is available through OCC input pins.

both — Combination of internal and external.

- capture_window_size : *int* ;

A property that specifies the maximum possible number of clock pulses the OCC can generate. During pattern generation, the OCC can be configured to produce a smaller number of pulses than this specified value. The default is 3, and the range is 2 to 6.

- type : standard | parent | child ;

A property that specifies what type of controller to build.

Choose one from the following:

standard — Request the creation of a standard OCC which has the ability to inject the shift clock and allows controlling which clock pulse goes through based on the control flip-flops. The fast_clock input can be free-running asynchronous clocks. See “[Fast Capture Mode Operation](#)” in the *Tessent Scan and ATPG User’s Manual* for a schematic drawing of this OCC type.

child — Request the creation of a child OCC which does not include a multiplexer along the clock path to inject the shift clock. The child OCC lets the clock go through when scan_en is high. When scan_en is low, the clock input pulses for a constant number of pulses and the child OCC lets zero or more pulses go through based on the value of its control flip-flops. Typically, an OCC of type parent is at the base of the clock sourcing the clock input of the child OCC. See “[Child OCC On-Chip Controller Logic Schematic](#)” in the *Tessent Scan and ATPG User’s Manual* for a schematic drawing of this OCC type.

parent — Request the creation of a parent OCC which has the ability to operate in standard mode or in parent mode. In parent mode, the OCC injects the shift clock at

the base of the clock in shift mode. When `scan_en` is low, it lets a programmable constant set of clock pulses go through. The parent OCC typically feeds the clock input of child OCC which are the ones that enable the scan-based test pattern generation tool to control which clock pulses go through or not. See “[Parent On-Chip Controller Logic Schematic](#)” in *Tessent Scan and ATPG User’s Manual* for a schematic drawing of this OCC type.

- `internal_clock_gater` : `on | off | auto` ;

A property that specifies if the OCC of type “child” has the clock gater inside the OCC module or if it is to only output the enable signal to be used by a clock gater already present at the base of the clock tree. It can only be set to off when type is “child”. When left at the default value of “auto”, it is interpreted as to mean off when the `clock_intercept_node` property is left empty and the `clock_enable_pin` property refers to the name of a pin.

- `shift_only_mode` : `on | off | auto` ;

A property that specifies the addition of a new port on the OCC controller that is used to activate the shift only mode in the OCC. The default is auto.

This property is “on” if you added the `ext_ltest_en` DFT signal using the [add_dft_signals](#) command. Otherwise, this property is “off”.

For more information, refer to “[Shift Mode](#)” in the *Tessent Scan and ATPG User’s Manual*.

- `kill_clock_mode` : `on | off | auto` ;

A property that specifies the addition of a new port on the OCC controller that is used to activate the clock gater on the fast_clock path such that the clock can be gated off during functional mode. The value of the `kill_clock_en` port is a don’t care when the OCC is active.

When the DFT signal `occ_kill_clock_en` is added, the auto value is interpreted as On, and the `kill_clock_en` port is automatically sourced by the `occ_kill_clock_en` DFT signal making it easy to activate this feature using the [set_static_dft_signal_values](#) command.

Use this property to re-use the clock gater inside the OCC during functional mode to kill the functional clock. You can also use this property during hierarchical test mode. When running the internal logic test modes on a core, you can save power inside the other cores that are not running by disabling their clocks using the following command and switch:

```
set_static_dft_signal_values occ_kill_clock_en 1 \
    -instances inactive_core_instances
```

To achieve the same results during MemoryBIST patterns, you use the `DftControlSettings` wrapper.

- `include_clocks_in_icl_model` : `on | off` ;

An optional property used to specify if the input and output clock ports as well as the optional TCK multiplexer to be described in the ICL model of the OCC. A TCK multiplexer is inserted inside the OCC when the `ijtag_host_node` property is not specified to none. The TCK mux must be described in ICL if you want to use the `tck_clock_only` property in Patterns file and the TCK clock is injected with the OCC controller.

- `leaf_instance_name : instance_name ;`

A property that specifies the leaf instance name for the clock controller.

Interface

This wrapper specifies the port names on the interface of the OCC module. When a particular property is not used, the tool default port naming will be used.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Interface {
            scan_en      : port_name ; // default: scan_en
            capture_en   : port_name ; // default: capture_en
            slow_clock   : port_name ; // default: slow_clock
            fast_clock   : port_name ; // default: fast_clock
            clock        : port_name ; // defualt: clock
            clock_out    : port_name ; // default: clock_out
            clock_en_out : port_name ; // default: clock_en_out
            scan_in      : port_name ; // default: scan_in
            scan_out     : port_name ; // default: scan_out
            clock_sequence: port_name ; // default: clock_sequence[%d]
            IJtagScanInterface {
            }
            StaticExternalControls {
            }
        }
    }
}
```

Description

This wrapper specifies the port names on the interface of the OCC module. When a particular property is not used, the tool default port naming will be used.

Arguments

- **scan_en : *port_name* ;**
Scan enable port name. The default is scan_en.
- **capture_en : *port_name* ;**
Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is capture_en. The default connection for capture_en is tie-0.
In LPCT usage, this tied pin is driven exclusively by the LPCT controller. In LBIST usage, this pin is multiplexed between LBIST capture enable during LBIST mode and inverted scan enable during ATPG mode.
- **slow_clock : *port_name* ;**
Slow clock input. This port provides the shift clock during shift mode and clock source during slow capture mode. In LPCT or LBIST applications, this is connected to a free running clock. The default is slow_clock. This port is not present when the type property is set to child.

- `fast_clock : port_name ;`
Fast clock input. This port is connected to the functional clock source and provides the source clock during fast capture mode. The default is `fast_clock`. This port is not present when the type property is set to child.
- `clock : port_name;`
A clock input that only exist when the type property is set to “child”. This clock carries the shift clock durign shift mode and the capture clock during capture mode. An OCC of type parent is typically used at the source of the clock feeding the OCC of type child.
- `clock_out : port_name ;`
Test clock output pin that delivers shift clock during shift and programmable capture clock during capture. The default is `clock_out`.
- `clock_en_out : port_name;`
A port that is only present for OCC of type “child”. It is used to control the `test_en` port of a clock gater situated at the based of the clock tree. This output is typically only used when the `internal_clock_gater` property is off.
- `scan_in : port_name ;`
Scan input for the shift register inside OCC. This pin will be declared as a scan sub-chain input for scan insertion. The default is `scan_in`.
- `scan_out : port_name ;`
Scan output for the shift register inside OCC. This pin will be declared as a scan sub-chain output for scan insertion. The default is `scan_out`.
- `clock_sequence : port_name ;`
Used for LogicBIST non-ATPG control of the OCC. The default is `clock_sequence[%d]`.

IjtagScanInterface

This wrapper specifies the names of ijtag interface ports of the OCC module. This wrapper is used only when the OCC uses IJTAG.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Interface {
            IjtagScanInterface {
                tck           : port_name      ; // default: ijtag_tck
                reset         : port_name     ; // default: ijtag_reset
                select        : port_name     ; // default: ijtag_sel
                capture_en   : port_name     ; // default: ijtag_ce
                shift_en     : port_name     ; // default: ijtag_se
                update_en    : port_name     ; // default: ijtag_ue
                scan_in      : port_name     ; // default: ijtag_si
                scan_out     : port_name     ; // default: ijtag_so
                reset_polarity: active_polarity; // legal : active_high
                                         // active_low
            }
        }
    }
}
```

Description

This wrapper specifies the names of ijtag interface ports of the OCC module. This wrapper is used only when the OCC uses IJTAG.

Arguments

- **tck : *port_name* ;**
IJTAG control signal. The default is ijtag_tck.
- **reset : *port_name* ;**
IJTAG control signal. The default is ijtag_reset.
- **select : *port_name* ;**
IJTAG control signal. The default is ijtag_sel.
- **capture_en : *port_name* ;**
IJTAG control signal. The default is ijtag_ce.
- **shift_en : *port_name* ;**
IJTAG control signal. The default is ijtag_se.
- **update_en : *port_name* ;**
IJTAG control signal. The default is ijtag_ue.

- `scan_in : port_name ;`
Scan interface pins for the IJTAG scan path that will be connected to the specified IJTAG host scan interface. The default is `ijtag_si`.
- `scan_out : port_name ;`
Scan interface pins for the IJTAG scan path that will be connected to the specified IJTAG host scan interface. The default is `ijtag_so`.
- `reset_polarity : active_polarity ;`
Active state of the `ijtag` reset signal, to be used during IJTAG network insertion. This specifies only the active state inside the OCC. The default is `active_low`.

StaticExternalControls

This wrapper specifies the names of static signals when ijtag is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Interface {
            StaticExternalControls {
                test_mode : port_name; //test_mode
                fast_capture_mode : port_name; //fast_capture_mode
                parent_mode : port_name; //parent_mode
                capture_cycle_width : port_name; //capture_cycle_width[%d]
                static_clock_control_mode : port_name; //static_clock_control_mode
                shift_only_mode : port_name; //shift_only_mode
                kill_clock_en : port_name; //kill_clock_en
            }
        }
    }
}
```

Description

This wrapper specifies the names of static signals when ijtag is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Arguments

- **test_mode : *port_name* ;**
Signal to specify non-Functional mode. The default value is test_mode. This port is only created when the [ijtag_host_interface](#) property is set to “none”.
- **fast_capture_mode : *port_name* ;**
Signal to choose between fast and slow capture modes. The source of the programmable clock output will be the fast_clock OCC input when this signal is 1 or the slow_clock OCC input when this signal is 0. The default value is fast_capture_mode. This port is only created when the [ijtag_host_interface](#) property is set to “none” and the [type](#) property is “standard” or “parent”.
- **parent_mode : *port_name* ;**
Signal used to enable the parent mode of the controller. The default value is parent_mode. This port is only created when the [ijtag_host_interface](#) property is set to “none” and the [type](#) property is “parent”.
- **capture_cycle_width : *port_name* ;**
Provides pattern generation time configurability of the capture width size up to the maximum specified. This is a vector port with log2(capture_width_size) bits. The default value is “capture_cycle_width”. This port is only created when the [ijtag_host_interface](#) property is set to “none”.

- static_clock_control_mode : *port_name* ;

Control signal to decide whether the programmable clock sequence is under ATPG control or not. When this pin is 0, the clock sequence is shifted in as part of scan loading. When this pin is 1, the clock sequence is parallel loaded from the OCC clock sequence. The default value is static_clock_control_mode. This port is only created when the [ijtag_host_interface](#) property is set to “none”.

- shift_only_mode : *port_name* ;

Signal used to enable the [shift_only_mode](#) port on the OCC controller. The default value is “shift_only_mode”. The port is only created when the [OCC](#) wrapper [shift_only_mode](#) property is set to or interpreted as “on”.

- kill_clock_en : *port_name* ;

Signal used to enable the [kill_clock_mode](#) port on the OCC controller that is used to activate the clock gater on the fast_clock path such that the clock can be gated off during functional mode. The default value is “kill_clock_en”. The port is only created when the [OCC](#) wrapper [kill_clock_mode](#) property is set to or interpreted as “on”.

Connections

This wrapper specifies the design port or pin to which the OCC pins should be connected. When specified, this globally overrides the default connection specified for this port type. These are the global specifications.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Connections {
            scan_en      : port_pin_name | DftSignal(scan_en) | ;
                           OptionalDftSignal(scan_en);
            capture_en   : port_pin_name | 0;
            slow_clock   : port_pin_name | DftSignal(shift_capture_clock) | ;
                           OptionalDftSignal(shift_capture_clock);
            clock_sequence : port_pin_constant_name, ...; // default: 0
            StaticExternalControls {
            }
        }
    }
}
```

Description

This wrapper specifies the design port or pin to which the OCC pins should be connected. When specified, this globally overrides the default connection specified for this port type. These are the global specifications.

Arguments

- **scan_en** : *port_pin_name* | *DftSignal(scan_en)* | *OptionalDftSignal(scan_en)*;
An optional property used to specify a source node to connect the *scan_en* port to.
The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.
When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify *OptionalDftSignal(scan_en)* or *DftSignal(scan_en)* as the value, and it will be interpreted to request a connection to the DFT signal *scan_en* as it was added using the [add_dft_signals](#) command. When using *OptionalDftSignal(scan_en)*, you will not get an error if the DFT signal *scan_en* was not added; a connection to the port called “*scan_en*” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error. When the property is unspecified, its value defaults to *OptionalDftSignal(scan_en)*.

- `capture_en : port_pin_name ;`

Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is 0.

- `slow_clock : port_pin_name ;`

An optional property used to specify a source node to connect the slow_clock port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the [set_design_level](#) command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify OptionalDftSignal(shift_capture_clock) or DftSignal(shift_capture_clock) as the value, and it will be interpreted to request a connection to the DFT signal shift_capture_clock as it was added using the [add_dft_signals](#) command. When using OptionalDftSignal(shift_capture_clock), you will not get an error if the DFT signal shift_capture_clock was not added; a connection to the port called “slow_clock” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error. When the property is unspecified, its value defaults to OptionalDftSignal(shift_capture_clock).

- `clock_sequence : port_pin_constant_name, ...; // default: 0`

Used for LogicBIST non-ATPG control of the OCC. The default is 0. When a constant value is provided, you should provide a sequence of bit values to be connected. For example, when capture_window_size is equal to 3, provided value should be of form ‘111’, ‘110’, ‘001’.

StaticExternalControls

This wrapper specifies the names of static signals when ijtag is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Usage

```
DftSpecification(module_name,id) {
    OCC {
        Connections {
            StaticExternalControls {
                test_mode : port_pin_name | 0 | 1;
                fast_capture_mode : port_pin_name | 0 | 1;
                parent_mode : port_pin_name | 0 | 1;
                capture_cycle_width : port_pin_name | 0 | 1, ... ;
                static_clock_control_mode : port_pin_constant_name | 0 | 1;
                shift_only_mode : port_name ;
                    // default: DftSignalOrTiedLow(ext_ltest_en)
                kill_clock_en : port_name;
                    // default: DftSignal(occ_kill_clock_en)
            }
        }
    }
}
```

Description

This wrapper specifies the names of static signals when the [ijtag_host_interface](#) property is set to “none”. When the [ijtag_host_interface](#) property refers to an IJTAG host node, these signals are available as TDR bits inside the OCC.

Arguments

- **test_mode** : *port_pin_name* | 0 | 1;
Signal that is used to activate the OCC. When this signal is 0, the OCC is not activated and simply lets the *fast_clock*/clock input go through unaltered. The default is 0.
- **fast_capture_mode** : *port_pin_name* | 0 | 1;
Signal to choose between fast and slow capture modes. The source of the programmable clock output will be the *fast_clock* OCC input when this signal is 1 or the *slow_clock* OCC input when this signal is 0. The default value is 0. This port is not present for OCC of type “child”.
- **parent_mode** : *port_pin_name* | 0 | 1 ;
Signal used to activate the parent mode of an OCC of type “parent”. When equal to 0, the OCC operates in standard mode. The default value is 0.
- **capture_cycle_width** : *port_pin_name* | 0 | 1, ... ;
Provides pattern generation time configurability of the capture width size up to the maximum specified. Constant throughout the test, that is ATPG cannot change it on a per pattern basis.

- static_clock_control_mode : *port_pin_constant_name* | 0 | 1 ;

Control signal to decide whether the programmable clock sequence is under ATPG control or not. When this pin is 0, the clock sequence is shifted in as part of scan loading. When this pin is 1, the clock sequence is parallel loaded from the OCC external_clock_control pins. The default is 0. This port only exists when the [static_clock_control](#) property is set to external or both.

- shift_only_mode : *port_name* ;

DFT signal used to activate the [shift_only_mode](#). The default connection is DftSignalOrTiedLow([ext_ltest_en](#)).

- kill_clock_en : *port_name* ;

DFT signal used to activate the [kill_clock_mode](#) such that the clock gater on the fast_clock path such that the clock can be gated off during functional mode. The default is DftSignal([occ_kill_clock_en](#)).

Controller

This wrapper specifies the connections to the controller.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Controller(id) {
            clock_intercept_node      : port_pin_name ;
            clock_enable_pin          : pin_name;
            clock_enable_pin_polarity : active_high | active_low | auto;
            parent_instance           : instance_name ;
            leaf_instance_name        : instance_name ;
            type                      : standard | child | parent;
            internal_clock_gater     : on | off | auto;
            shift_only_mode           : on | off | auto;
            kill_clock_mode           : on | off | auto;
            Connections {
            }
        }
    }
}
```

Description

This wrapper specifies the connections to the controller.

Arguments

- **clock_intercept_node** : *port_pin_name* ;

A property that specifies the design port or pin to which the OCC clock output pin will be attached. The value of the “has_functional_source” attribute of the port or pin specified by this property impacts the connectivity of the OCC fast clock port. See the `fast_clock` property description in “OCC/Controller/[Connections](#)”.

- **clock_enable_pin** : *pin_name*;

A property that specifies the `test_en` pin of a clock gater situated at the base of the clock tree which the OCC of type `child` uses to control the clock during scan-based test pattern generation. This property can only be used when the `type` property is set to “`child`”. When the `clock_enable_pin` property is used and the `clock_intercept_node` property is left blank, the `internal_clock_gater` is inferred to `off` when it has a value of “`auto`”.

- **clock_enable_pin_polarity** : `active_high` | `active_low` | auto;

This property controls the active polarity of the `clock_en_port`. You only need to specify this property when you are creating the OCC without doing the insertion ([process_dft_specification](#) is used with the `-no_insertion` switch) and the `clock_enable_pin` is not pointing to the `test_en` pin of an integrated clock gater cell. When it is pointing to the `test_en` pin of an integrated clock gater cell, the active polarity is automatically extracted from the cell information.

- `parent_instance : instance_name ;`

This property specifies the design instance in which the OCC is instantiated. By default, the OCC is placed at the same hierarchical level as the `clock_intercept_node`.

- `leaf_instance_name : instance_name ;`

This property inherits its value from the OCC wrapper `leaf_instance_name` property if not specified locally.

- `type : standard | child | parent;`

This property inherits its values from the `../type` property when it is not specified locally. Its meaning is identical to how it is described in the `../type` property section.

- `internal_clock_gater : on | off | auto;`

This property inherits its values from the `../internal_clock_gater` property when it is not specified locally. Its meaning is identical to how it is described in the `../internal_clock_gater` property section.

- `shift_only_mode : on | off | auto;`

This property inherits its value from the `../shift_only_mode` property when it is not specified locally.

- `kill_clock_mode : on | off | auto;`

This property inherits its value from the `../kill_clock_mode` property when it is not specified locally.

Connections

This wrapper specifies the design port or pin to which the OCC pins should be connected.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Controller(id) {
            Connections {
                scan_en      : port_pin_name | DftSignal(scan_en) |;
                OptionalDftSignal(scan_en);
                capture_en   : port_pin_name | 0;
                slow_clock   : port_pin_name | DftSignal(shift_capture_clock) |
                OptionalDftSignal(shift_capture_clock)
                fast_clock   : port_pin_name ;
                clock        : port_pin_name ;
                clock_sequence: port_pin_constant_name, ... ; // default: 0
                StaticExternalControls {
                }
            }
        }
    }
}
```

Description

This wrapper specifies the design port or pin to which the OCC pins should be connected.

Arguments

- `scan_en : port_pin_name ;`

An optional property used to specify a source node to connect the `scan_en` port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the `set_design_level` command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(scan_en)` or `DftSignal(scan_en)` as the value, and it will be interpreted to request a connection to the DFT signal `scan_en` as it was added using the `add_dft_signals` command. When using `OptionalDftSignal(scan_en)`, you will not get an error if the DFT signal `scan_en` was not added; a connection to the port called “`scan_en`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value is inherited from the [../Connections/scan_en](#) property which itself defaults to `OptionalDftSignal(scan_en)` when unspecified.

- `capture_en : port_pin_name ;`

Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is 0.

- `slow_clock : port_pin_name ;`

An optional property used to specify a source node to connect the shift_capture_clock port to.

The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the `set_design_level` command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it will be created as an input port during insertion.

When design level is “chip”, specified port names must already be connected to a pad cell. The connection will automatically be made to the internal side of the input pad buffer. You can specify `OptionalDftSignal(shift_capture_clock)` or `DftSignal(shift_capture_clock)` as the value, and it will be interpreted to request a connection to the DFT signal `shift_capture_clock` as it was added using the `add_dft_signals` command. When using `OptionalDftSignal(shift_capture_clock)`, you will not get an error if the DFT signal `shift_capture_clock` was not added; a connection to the port called “`slow_clock`” will be made instead and the port will be created if not present unless design level is equal to “chip” in which case you will get an error.

When the property is unspecified, its value is inherited from the `../Connections/shift_capture_clock` property which itself defaults to `OptionalDftSignal(shift_capture_clock)` when unspecified.

- `fast_clock : port_pin_name ;`

This property is used to specify a connection for the `fast_clock` input port of the OCC. When making this connection, the tool considers the value of the “`has_functional_source`” attribute of the port or pin specified by the `clock_intercept_node` and whether or not the `fast_clock` property is specified. There are four cases:

- Case 1 — The specified `clock_intercept_node` has a functional source, meaning its “`has_functional_source`” attribute set to true, and the `fast_clock` property is not specified. The OCC intercepts the specified `clock_intercept_node` so that the specified port/pin drives the OCC `fast_clock` port and the OCC `clock_output` drives all the sinks previously driven by the specified pin.
- Case 2 — The port or pin specified by `clock_intercept_node` has no functional source and the `fast_clock` property is specified. The OCC is inserted such that its `fast_clock` pin is connected to the port/pin specified by the `fast_clock` property and its `clock_out` pin is connected to the port/pin specified by the `clock_intercept_node` property.
- Case 3 — The specified `clock_intercept_node` has its “`has_functional_source`” attribute set to true, and the `fast_clock` property is specified. The OCC is inserted so

that its fast_clock pin is connected to the port/pin specified by the fast_clock property and the OCC clock_out pin drives the logic previously connected to the port/pin specified by the clock_intercept_node property.

- Case 4 — The clock_intercept_node port does not have a functional source and fast_clock is not specified. In this case, the tool reports an error.
- clock : port_pin_name ;

This property is used to specify a connection for the clock input port of the OCC which only exists on OCC of type “child”. This port is connected to a functional clock source and provides the source clock during both shift and capture mode. Used only when the clock_intercept_node property points to a pin that has no functional source (specifically, its has_functional_source attribute is false).

- clock_sequence : port_pin_constant_name, ... ;

Used for LogicBIST non-ATPG control of the OCC. The default is 0. When a constant value is provided, you should provide a sequence of bit values to be connected. For example, when capture_window_size is equal to 3, provided value should be of form ‘111’, ‘110’, ‘001’.

Examples

Example 1

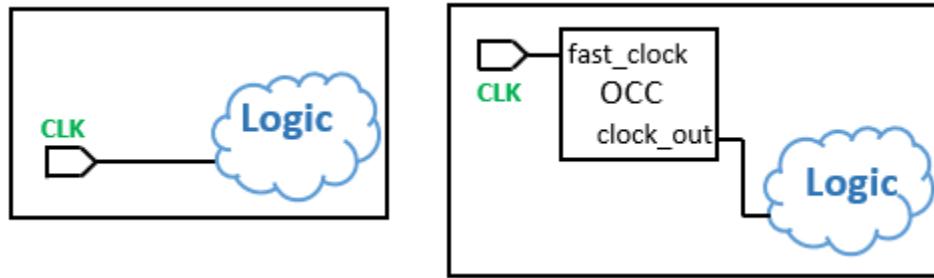
This example shows the case when the specified clock_intercept_node has a functional source and the fast_clock option is not specified for the OCC controller.

The OCC/Controller property clock_intercept_node specifies the CLK port (highlighted in green). The OCC/Controller/Connections fast_clock property is not used.

```
DftSpecification(design1, gates) {
    OCC {
        Controller(NX1) {
            clock_intercept_node: CLK;
        }
    }
}
```

Figure 10-54 shows a design before and after OCC insertion. The OCC is inserted so that port specified by clock_intercept_node, CLK (highlighted in green), is connected to the OCC fast_clock port and the OCC clock_out port is connected to the logic previously connected to CLK.

Figure 10-54. Case 1 — clock_intercept_node Has Functional Source and fast_clock Not Specified



Example 2

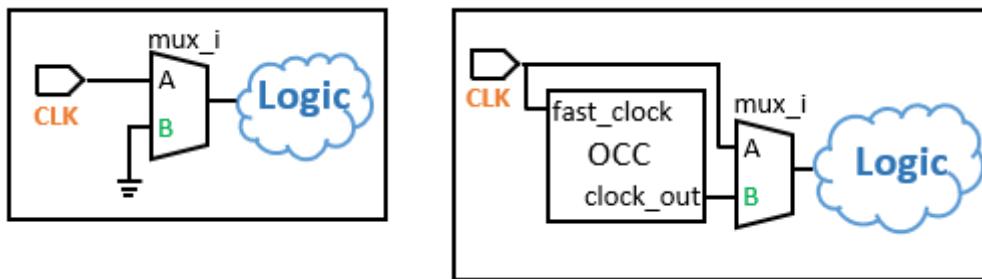
This example shows case 2 when the specified controller_intercept_node does not have a functional source and the fast_clock option is specified for the OCC controller. One example for this case is when you are using a MUX outside the OCC to switch between the functional clock and the test clock.

The specification shows that the clock_intercept_node is a pin on MUX, mux_i/B (highlighted in green), which is tied low. It also shows that the OCC/Controller/Connections fast_clock property is set to CLK (highlighted in orange).

```
DftSpecification(design1, gates) {
    OCC {
        Controller(NX1) {
            clock_intercept_node: mux_i/B;
            Connections {
                fast_clock : CLK;
            }
        }
    }
}
```

Figure 10-55 shows that, before OCC insertion, CLK was connected to mux_i/A and mux_i/B is tied low. After insertion, the CLK port (highlighted in orange) is still connected to mux_i/A. It is also connected to the OCC fast_clock port. The OCC clock_out pin is connected to mux_i/B (highlighted in green).

Figure 10-55. No Functional Source and fast_clock Specified



Note

 An error is reported if the clock_intercept_node does not have a functional source and fast_clock is not specified.

Example 3

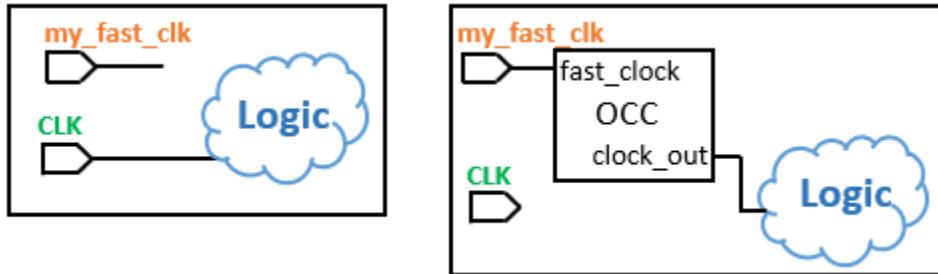
This example shows the case when the specified clock_intercept_node has a functional source and the fast_clock option is specified for the OCC controller.

The OCC/Controller property clock_intercept_node specifies the CLK port (highlighted in green). The OCC/Controller/Connections fast_clock property specifies the my_fast_clock port (highlighted in orange).

```
DftSpecification(design1, gates)  {
    OCC {
        Controller(NX1) {
            clock_intercept_node: CLK;
            Connections {
                fast_clock : my_fast_clock;
            }
        }
    }
}
```

Figure 10-56 shows that before OCC insertion the clock_intercept_node, CLK (highlighted in green) is connected to logic. It also shows the fast_clock specified port, my_fast_clock (highlighted in orange). After OCC insertion my_fast_clock is connected to the fast_clock input of the OCC and the clock_out port of the OCC drives the logic previously connected directly to CLK.

Figure 10-56. Case 3 - Functional Source and fast_clock Specified



StaticExternalControls

This wrapper specifies the names of static signals when ijtag is not used.

Usage

```
DftSpecification(module_name, id) {
    OCC {
        Controller(id) {
            Connections {
                StaticExternalControls {
                    test_mode : port_pin_name | 0 | 1;
                    fast_capture_mode : port_pin_name | 0 | 1;
                    parent_mode : port_pin_name | 0 | 1;
                    capture_cycle_width : port_pin_name | 0 | 1, ... ;
                    static_clock_control_mode : port_pin_constant_name | 0 | 1;
                    shift_only_mode : port_name;
                    kill_clock_en // default: DftSignalOrTiedLow(ext_ltest_en)
                      : port_name;
                    // default: DftSignal(occ_kill_clock_en)
                }
            }
        }
    }
}
```

Description

This wrapper specifies the names of static signals when ijtag is not used.

Arguments

- `test_mode : port_pin_name ;`

This property inherits its value from the `../Connections/StaticExternalControls/test_mode` property when unspecified. It has the exact same usage as described in the `../Connections/StaticExternalControls/test_mode` section. You specify it here if you want a different connection for this controller.

- `fast_capture_mode : port_pin_name ;`

This property inherits its value from the `../Connections/StaticExternalControls/fast_capture_mode` property when unspecified. It has the exact same usage as described in the `../Connections/StaticExternalControls/fast_capture_mode` section. You specify it here if you want a different connection for this controller.

- `parent_mode : port_pin_name ;`

This property inherits its value from the `../Connections/StaticExternalControls/parent_mode` property when unspecified. It has the exact same usage as described in the `../Connections/StaticExternalControls/parent_mode` section. You specify it here if you want a different connection for this controller.

- capture_cycle_width : *port_pin_name*, ... ;

This property inherits its value from the ../../Connections/StaticExternalControls/[capture_cycle_width](#) property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/[capture_cycle_width](#) section. You specify it here if you want a different connection for this controller.

- static_clock_control_mode : *port_pin_constant_name* ;

This property inherits its value from the ../../Connections/StaticExternalControls/[static_clock_control_mode](#) property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/[static_clock_control_mode](#) section. You specify it here if you want a different connection for this controller.

- shift_only_mode : *port_name* ;

This property inherits its value from the ../../Connections/StaticExternalControls/[shift_only_mode](#) property and has the exact same usage. You specify it here if you want a different connection for this controller. The default is DftSignalOrTiedLow(ext_ltest_en).

- kill_clock_en : *port_name* ;

This property inherits its value from the ../../Connections/StaticExternalControls/[kill_clock_en](#) property and has the exact same usage. You specify it here if you want a different connection for this controller. The default DftSignal(occ_kill_clock_en).

RtlCells

A wrapper that is used to specify the name of the ports to use when creating RTL cells.

Usage

```
DftSpecification(module_name,id) {
    RtlCells {
        And2|Or2|ClkAnd2|ClkOr2 {
            input0 : string ; // default: a
            input1 : string ; // default: b
            output : string ; // default: y
        }
        Buf|Inv|ClkBuf|ClkInv {
            input : string ; // default: a
            output : string ; // default: y
        }
        Mux2|ClkMux2 {
            input0 : string ; // default: a
            input1 : string ; // default: b
            select : string ; // default: s
            output : string ; // default: y
        }
        ClkGateAnd|ClkGateOr {
            clock_in : string ; // default: clk
            functional_enable : string ; // default: fe
            test_enable : string ; // default: te
            clock_out : string ; // default: clkg
        }
        PosedgeDff|NegedgeDff {
            data_in : string ; // default: d
            clock_in : string ; // default: clk
            data_out : string ; // default: q
        }
        PosedgeDffReset|NegedgeDffReset {
            reset : string ; // default: rn
            reset_polarity : string ; // active_high | active_low
            data_in : string ; // default: d
            clock_in : string ; // default: clk
            data_out : string ; // default: q
        }
        PosedgeDffSet|NegedgeDffSet {
            set : string ; // default: sn
            set_polarity : string ; // active_high | active_low
            data_in : string ; // default: d
            clock_in : string ; // default: clk
            data_out : string ; // default: q
        }
        PosedgeSynchronizer {
            data_in : string ; // default: d
            clock_in : string ; // default: clk
            data_out : string ; // default: q
        }
        PosedgeSynchronizerReset {
            reset : string ; // default: rn
            reset_polarity : string ; // active_high | active_low
            data_in : string ; // default: d
            clock_in : string ; // default: clk
            data_out : string ; // default: q
        }
    }
}
```

Description

A wrapper that is used to specify the name of the ports to use when creating RTL cells. A set of RTL cells is created when the `use_rtl_cells` property of the **DftSpecification** wrapper defaults to on or is specified as on. The list of cells is dependent on the need of the instruments being created, and the list requested in the `force_creation_of_rtl_cells : cell_type, ... ;` command.

Arguments

- `And2|Or2|ClkAnd2|ClkOr2/input0 : string ;`
A property that specifies the name of the first input of the And or Or cell. For the Clock version, this input is the input to which the clock signal is connected. The default string is “a”.
- `And2|Or2|ClkAnd2|ClkOr2/ input1 : string ;`
A property that specifies the name of the second input of the And or Or cells. For the Clock version, this input is the input to which the data signal is connected. The default string is “b”.
- `And2|Or2|ClkAnd2|ClkOr2/ output : string ;`
A property that specifies the name of the output of the And or Or cells. The default string is “y”.
- `Buf|Inv|ClkBuf|ClkInv/input : string ;`
A property that specifies the name of the input of the Buffer or Inverter cells. The default string is “a”.
- `Buf|Inv|ClkBuf|ClkInv/output : string ;`
A property that specifies the name of the output of the Buffer or Inverter cells. The default string is “y”.
- `Mux2|ClkMux2/input0 : string ;`
A property that specifies the name of input0 of the multiplexer cells. The default string is “a”.
- `Mux2|ClkMux2/input1 : string ;`
A property that specifies the name of input1 of the multiplexer cells. The default string is “b”.
- `Mux2|ClkMux2/select : string ;`
A property that specifies the name of the select input of the multiplexer cells. The default string is “s”.
- `Mux2|ClkMux2/output : string ;`
A property that specifies the name of the output of the multiplexer cells. The default string is “y”.

- **ClkGateAnd|ClkGateOr/clock_in : string ;**
A property that specifies the name of the clock input of the clock gating cells. The default string is “clk”.
- **ClkGateAnd|ClkGateOr/functional_enable : string ;**
A property that specifies the name of the functional enable input of the clock gating cells. The default string is “fe”.
- **ClkGateAnd|ClkGateOr/test_enable : string ;**
A property that specifies the name of the test enable input of the clock gating cells. The default string is “te”. The input is used to disable the gating during scan test.
- **ClkGateAnd|ClkGateOr/clock_out : string ;**
A property that specifies the name of the gated clock output of the clock gating cells. The default string is “clkg”.
- **PosedgeDff|NededgeDff/data_in : string ;**
A property that specifies the name of the data input of the DFF cells. The default is “d”.
- **PosedgeDff|NededgeDff/clock_in : string ;**
A property that specifies the name of the clock input of the DFF cells. The default is “clk”.
- **PosedgeDff|NededgeDff/data_out : string ;**
A property that specifies the name of the data output of the DFF cells. The default is “q”.
- **PosedgeDffReset | NededgeDffReset/reset : string ;**
A property that specifies the name of the reset of the DFF cells. The default is “rn”.
- **PosedgeDffReset | NededgeDffReset/reset_polarity : string ;**
A property that specifies the name of the reset polarity of the DFF cells. The default is “active_low”.
- **PosedgeDffReset | NededgeDffReset/data_in : string ;**
A property that specifies the name of the data input of the DFF cells. The default is “d”.
- **PosedgeDffReset | NededgeDffReset/clock_in : string ;**
A property that specifies the name of the clock input of the DFF cells. The default is “clk”.
- **PosedgeDffReset | NededgeDffReset/data_out : string ;**
A property that specifies the name of the data output of the DFF cells. The default is “q”.
- **PosedgeDffSet | NededgeDffSet/set : string ;**
A property that specifies the name of the set of the DFF cells. The default is “sn”.
- **PosedgeDffSet | NededgeDffSet/set_polarity : string ;**
A property that specifies the name of the set polarity of the DFF cells. The default is “active_low”.

- PosedgeDffSet | NegedgeDffSet/*data_in* : *string* ;
A property that specifies the name of the data input of the DFF cells. The default is “d”.
- PosedgeDffSet | NegedgeDffSet/*clock_in* : *string* ;
A property that specifies the name of the clock input of the DFF cells. The default is “clk”.
- PosedgeDffSet | NegedgeDffSet/*data_out* : *string* ;
A property that specifies the name of the data output of the DFF cells. The default is “q”.
- PosedgeSynchronizer/*data_in* : *string* ;
A property that specifies the name of the data input of the synchronizer cells. The default is “d”.
- PosedgeSynchronizer/*clock_in* : *string* ;
A property that specifies the name of the clock input of the synchronizer cells. The default is “clk”.
- PosedgeSynchronizer/*data_out* : *string* ;
A property that specifies the name of the data output of the synchronizer cells. The default is “q”.
- PosedgeSynchronizerReset/*reset* : *string* ;
A property that specifies the name of the reset of the synchronizer reset cells. The default is “rn”.
- PosedgeSynchronizerReset/*reset_polarity* : *string* ;
A property that specifies the name of the reset polarity of the synchronizer reset cells. Choose either active_high or active_low. The default is “active_low”.
- PosedgeSynchronizerReset/*data_in* : *string* ;
A property that specifies the name of the data input of the synchronizer reset cells. The default is “d”.
- PosedgeSynchronizerReset/*clock_in* : *string* ;
A property that specifies the name of the clock input of the synchronizer reset cells. The default is “clk”.
- PosedgeSynchronizerReset/*data_out* : *string* ;
A property that specifies the name of the data output of the synchronizer reset cells. The default is “q”.

Examples

This example specifies the port name for the clock multiplexer cell.

```
DftSpecification(modA,rtl) {
    RtlCells {
        ClockMux {
            input0 : I0 ;
            input1 : I1 ;
            select : S0 ;
            output : O ;
        }
    }
}
```

PatternsSpecification Configuration Syntax

This chapter describes the configuration data syntax used inside Tesson Shell to specify patterns to be applied to DFT components that are inserted into a design.

This chapter covers the [BoundaryScan](#), [MemoryBist](#), [MemoryBisr](#), [LogicBist](#), and [SdfInfo](#) wrappers as they exist inside the PatternsSpecification wrapper. Using this syntax, it is possible to specify a wide variety of patterns specifications, in a single wrapper or in multiple PatternsSpecification wrappers, to generate verification test benches used for simulation and tester patterns used for chip manufacturing.

PatternsSpecification	3570
Patterns	3576
SimulationOptions	3580
AdvancedOptions	3585
TestStep	3590
ICLNetworkVerify	3655
ProcedureStep	3659
EnableGroupInfo	3669
SdfInfo	3671
LoadBoardInfo	3674
TesterInterface	3676
Loopbacks	3679
ACLoopbacks	3681
InSystemTest	3683

PatternsSpecification

Defines a series of Patterns wrapper for a given view of a design.

Usage

```
PatternsSpecification(design_name, design_id, pattern_id) {
    usage                      : simulation | manufacturing_test ;
    manufacturing_patterns_format : format, ... ; // default: Stil
    compress_pattern_files       : boolean // default: on
    SimulationOptions {
        testbench_module_name     : format_string ;
        logic_bist_debug          : off | bist_registers_and_clock_verify |
                                    // setup_and_clock_verify |
                                    // setup_and_clock_verify_one_per_ncp;
        verify_internal_data_pins : boolean ; // default: off
    }
    AdvancedOptions {
        parameter_file           : file_path ;
        Parameters {
            parameter_name         : parameter_value ;
        }
        procfile_name             : file_path ;
        ireset_in_test_setup_proc : on | off ;
        no_initialization        : on | off ;
        ConstantPortSettings {
            port_name              : 0 | 1 ;
        }
    }
    Patterns(patterns_name) {
    }
    PatternsGroup(group_name) {
        AdvancedOptions {
            // same as ../AdvancedOptions
        }
        Patterns(patterns_name) {
        }
    }
    EnableGroupInfo(enable_group_info_id) {
    }
    SdfInfo(sdf_info_id) {
    }
    LoadBoardInfo (load_board_info_id) {
    }
    InSystemTest {
    }
}
```

Description

The PatternsSpecification wrapper defines a series of Patterns wrapper for a given view of a design. The Patterns wrapper can be declared directly at the top level or grouped into PatternsGroup wrappers.

The pattern specification uniquely identifies a design view of a design by using the *design_name* and *design_id* values. The [create_patterns_specification](#) command sets the

pattern_id value to signoff or manufacturing based on the usage arguments. The *pattern_id* can later be modified to create various PatternsSpecifications. For example, you can create a patterns specification targeted for wafer probe test, final test, and board level test. The number of tester channels and power available in each test setup typically varies and patterns need to be adjusted accordingly.

Arguments

- *design_name*

A string that identifies the PatternsSpecification for a specific design. This string is automatically populated by the [create_dft_specification](#) command to match the name of the current design. Refer to the description of the [process_patterns_specification](#) command to understand which PatternsSpecification gets processed.

- *design_id*

A string that identifies the PatternsSpecification for a specific *design_id* of the current design. If this string is missing from the “get_context -design_id” command and switch, it is automatically populated by the [create_patterns_specification](#) command with the value of the [tessent_design_id](#) attribute found in the top ICL module. Refer to the description of the [process_patterns_specification](#) command to understand which PatternsSpecification the tool processes.

- *pattern_id*

A string that distinguishes multiple PatternsSpecifications for a specific *design_name* and *design_id* from each other. The [create_patterns_specification](#) command populates this string with signoff and manufacturing but this string can be changed to any value to differentiate different patterns specifications from each other. For example, you can create a system_test version that is derived from the manufacturing one and include only the tests that can be operated through the TAP controller.

- usage : [simulation](#) | [manufacturing_test](#) ;

A property that specifies whether the PatternsSpecification is to be used to generate simulation test benches or manufacturing test patterns. The patterns are converted to Verilog test benches when the usage is specified as “simulation” and to the specified [manufacturing_patterns_format](#) when the usage is specified as “manufacturing_test”. Some settings are checked against this usage specification such as the “reduced_address_count” property in the [MemoryBist](#) wrapper.

- [manufacturing_patterns_format](#) : *format*, ... ;

A property that specifies one or more formats for the patterns files to generate when usage is specified as “manufacturing_test”.

The property supports the following formats:

- CTL
- FJTDL

- MITDL
- STIL — The default.
- STIL2005
- SVF — Because of inherent limitations of SVF, you *cannot* specify this format at the same time as other formats.
- TITDL
- TSTL2
- WGL
- WGL_LSI

For a description of these formats, see the description of the *format_switch* argument to the [write_patterns](#) command.

- `compress_patterns_file : on | off ;`

A property that specifies whether the patterns files are to be compressed when written to disk. When the property is specified to on, the file will have the extension “.gz” to signify that it is compressed.

- `testbench_module_name : format_string ;`

A property that specifies the top-level module name in the Verilog test bench file. You can use this property to give a different test bench module name for each test bench so that more than one test bench can be compiled in a single work area of a Verilog simulator at the same time.

The default is TB. The string can also include %d which will be replaced by the name of the current design, or %p which will be replaced by the name of the Patterns wrapper. Using a unique name for the test bench allows you to use a common signal name file to display signals in your simulator including alias name definitions you may have created. The [run_testbench_simulations](#) command uses a separate directory to compile each test bench so keeping the testbench_module_name as TB is optimal and recommended.

- `logic_bist_debug : off | bist_registers_and_clock_verify | setup_and_clock_verify | setup_and_clock_verify_one_per_ncp ;`

A property that causes the tool to provide extra data to help you debug LogicBIST simulation mismatches. Refer to “[Pattern Mismatch Debugging](#)” in the *Hybrid TK/LBIST User’s Manual* for details. The valid values are:

- off — The tool does not provide extra data for debugging.
- bist_registers_and_clock_verify — The simulation generates a mismatch showing when the PRPG, MISR, and low-power registers (if present) diverge from their expected values. This can help reduce debug time because it identifies the scan pattern at which the register(s) begin to diverge.

- `setup_and_clock_verify` — The tool creates a LogicBIST pattern that verifies that the controller is setup correctly, and then it applies 256 serial patterns to exercise the full NCP count range. During this test, the tool reduces the shift length to decrease the simulation runtime. Because of the reduced shift length, the MISR signature is not examined at the end of the pattern. If a clock does not generate the expected number of pulses for a given NCP, the tool generates a simulation mismatch.
- `setup_and_clock_verify_one_per_ncp` — The same as `setup_and_clock_verify`, except the tool only applies a single scan pattern for each NCP.
- `AdvancedOptions/parameter_file : file_path ;`
A property that specifies the name of a file containing parameters used to configure the `write_patterns` command. For a description of the available parameters, see the “[Parameter File Format and Keywords](#)” on page 3937.
- `AdvancedOptions/Parameters/parameter_name : parameter_value ;`
A wrapper that specifies parameter-value pairs used to configure the `write_patterns` command. A parameter specified in this wrapper overrides the same parameter specified in the specified `parameter_file`. For a description of the available parameters, see the “[Parameter File Format and Keywords](#)” on page 3937.
- `AdvancedOptions/procfile_name : file_path ;`
A property that specifies a procfile name that contains a `test_setup` and/or a `test_end` procedure.

The `file_path` must have one of the following formats:

- A simple file name with no directory path. In this case, the file must exist beside the ICL file.
- A null string. In this case, no procfile is used.
- An absolute path starting with a “/”.
- A relative path starting with “./” or “..”. In this case, the file must exist relative to the current working directory.
- `AdvancedOptions/ireset_in_test_setup_proc : on | off ;`
A property which, when set to “on”, will suppress the “iReset” being issued before the first pattern set generated for the Patterns wrapper being processed.
A `test_setup` procedure must be provided in the procfile specified by the “`procfile_name`” property in the same `AdvancedOptions` wrapper.
It is your responsibility to put appropriate events into the `test_setup` procedure that will reset the IJTAG network. The effect of the `ireset_in_test_setup_proc` being on is to add the `-ireset_in_test_setup_proc` to the `open_pattern_set` command for the first pattern set generated by for the Patterns wrapper.

- AdvancedOptions/no_initialization : on | off ;

A property that specifies creating the initialization cycle in the pattern file. The default is off.

If defined, the value of no_initialization property becomes the default value for the property in the following wrappers:

- PatternsSpecification/PatternsGroup/AdvancedOptions
- PatternsSpecification/Patterns/AdvancedOptions

This property maps directly to the [write_patterns](#) -noinitialization command and switch when the patterns are written out by [process_patterns_specification](#).

- AdvancedOptions/ConstantPortSettings/port_name : 0 | 1 ;

A data wrapper that specifies the ports that must be held constant throughout the entire pattern set. This list is typically empty and is not populated by the [create_patterns_specification](#) command. Input ports that were constrained prior to running extract_icl are automatically reported in the ICL file, and they are automatically forced accordingly when the [process_patterns_specification](#) command is called.

- PatternsGroup(*group_id*)

A wrapper that groups a set of Patterns wrappers. The group_id value is a unique string containing only letters, numbers, and underscores that identifies the group. This option makes it convenient to fold that group into the GUI when using the [add_config_tab](#) command. It also enables you to select that group in the GUI to validate or process all Patterns wrappers found inside the group.

You can also do this from the dofile using the [process_patterns_specification](#) command with the -config_object option referring to a PatternsGroup wrapper.

Examples

The following example shows a PatternsSpecification wrapper for module “MyDesign”, design_identifier “rtl”, and pattern_id set to signoff. The AdvancedOptions/Parameters wrapper is used to specify the SIM_PRECISION parameter. You typically do not need to set this parameter as it is automatically set to provide accuracy of each clock period to within 1%. For example, a Patterns wrapper using a clock with a period of 4.5 ns and 450 ps will automatically have a resolution of 1 ps such that the 450 ps half period of 225 ps can be expressed with more than 2 ps or 1% accuracy. It is shown here as an example in case you wanted to set a higher precision.

```
PatternsSpecification(MyDesign,rtl,signoff) {
    usage : simulation ;
    AdvancedOptions {
        Parameters {
            SIM_PRECISION : 1ps ;
        }
    }
}
```

Related Topics

[create_patterns_specification](#)
[process_patterns_specification](#)

Patterns

Defines a complete set of Patterns steps each of which is meant to be run as one entity during simulation or manufacturing test.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        tester_period : time ;
        tck_clock_only : on | off ;
        tck_off_state : 0 | 1 ;
        tck_ratio : ratio ;
        load_board_info : load_board_info_id ;
        timeplate : timeplate_name ;
        DftControlSettings {
            icl_inst_path : 1 | 0 ;
        }
        ClockPeriods {
            pin_name : tester | time ; // repeatable
        }
        AdvancedOptions {
        }
        SimulationOptions {
        }
        ICLNetworkVerify(id) { // mutually exclusive with ProcedureStep and
        } // and TestStep wrappers
        ProcedureStep(procedure_step_name) { //repeatable
        }
        TestStep(step_name) { //repeatable
        }
    }
}
```

Description

The Patterns wrapper defines a complete set of Patterns steps each of which is meant to be run as one entity during simulation or manufacturing test. When the PatternsSpecification/usage property is set to manufacturing_test, it is possible to split separate steps into multiple patterns file such that test program events such as VDD bumping or PMU measurements can be inserted in the middle of the patterns. See the AdvancedOptions/split_patterns_file property description in the [ProcedureStep](#) and [TestStep](#) wrappers for more information about patterns file splitting.

Arguments

- tester_period : *time* ;

A property that sets the tester period for the given pattern set. Clocks defined as “tester” in the ClockPeriods wrapper will operate at the specified tester period. The TCK clock operates at the specified tester period divided down by the specified tck_ratio property.

- tck_clock_only : on | off ;

A property that specifies that all instruments in this Patterns wrapper should run with a TCK clock.

Note

 The tck_clock_only property will only have an effect on Patterns containing MemoryBIST tests.

This property will instruct the IJTAG solver to find a way to sensitize clock muxes and OCC on the clock paths to inject a TCK source. If there is no way to inject TCK from a TCKPort, an error will be issued. The test can still be run at TCK if the traced functional ClockPorts are set as “tester_period” in the ClockPeriods wrapper. The patterns will then assume that TCK will be injected on those functional clock ports from outside the design.

The TCK injection needs to be present in the ICL descriptions of the OCC modules in order to be understood and used by the IJTAG solver. To make sure the OCC ICL is complete, you must set the OCC wrapper property [include_clocks_in_icl_model](#) to “on” before processing the DftSpecification with the [process_dft_specification](#) command.

- `tck_off_state : 0 | 1 ;`

A property that specifies the off value of the TCK clock. By default, the off value is 0. See the description of the [open_pattern_set](#) command for a description of the TCK waveform based on the `tck_off_state` and the `tck_ratio` values.

- `tck_ratio : ratio;`

A property that specifies the ratio of the period of TCK with respect to the tester period. By default, the ratio is 1. See the description of the [open_patterns_set](#) command for a description of the TCK waveform based on the `tck_off_state` and the `tck_ratio` values.

This value is set correctly by the [create_patterns_specification](#) command when it uses tester clocks to bring down the period of TCK to the appropriate value.

- `load_board_info : load_board_info_id ;`

A property used to reference a [LoadBoardInfo](#) wrapper. The tester interface and loopbacks specified in the pointed LoadBoardInfo wrapper will be considered in the patterns and the Verilog test bench will be created to reflect those if it is generated.

- `timeplate : timeplate_name ;`

A property that specifies the timeplate the tool uses to override the default timeplate assumed when generating the pattern. The timeplate is loaded from a procfile as specified from the PatternsSpecification/Patterns/[AdvancedOptions](#)/procfile_name property.

- `DftControlSettings/signal : 1 | 0 ;`

A data wrapper that specifies the values of static dft signals of type `global_dft_control` that were created during the DFT step using the [add_dft_signals](#) command. By default, all dft control signals of type `global_dft_control` are set to their reset value unless you explicitly specify them in the DftControlSettings wrapper.

To report the available static dft signals that have the `global_dft_control` type, use the “[report_static_dft_signal_settings -all](#)” command when the ICL is elaborated.

See [Example 2](#) to understand how this wrapper is auto populated by the `create_patterns_specification` command.

When the `create_patterns_specification` creates a Patterns wrapper, it will often automatically fill out the `DftControlSettings` wrapper with some DFT signals. For every active physical block instance, all DFT signals which were registered with a `-defaults_value_in_all_test` that is different than its `-reset_value` are set to their specified `-defaults_value_in_all_test` values. When creating memoryBIST patterns, the `nonscan_test` DFT signal is also set.

- `ClockPeriods/pin_name : tester | time ;`

A data wrapper that specifies the clocks that are to be started for the given pattern set. When the value “tester” is used for a given clock, the clock is started as a free-running synchronous clock operating at the tester period. When the value is of type time, the clock is started as an asynchronous clock with its period set to the specified time value. Those asynchronous clocks are present in the simulation test benches but only appear as annotations near the top of the patterns file when the usage is `manufacturing_test`. The test engineer is expected to start the asynchronous clock with tester-specific commands. This is an example annotation that appears in a stil file to report the asynchronous clocks:

```
Ann { * Begin_async_clocks * }
Ann { *   clk = period: 16.00ns * }
Ann { * End_async_clocks * }
```

If a controller used in the current Patterns wrapper issues an iClock command on its clock port and the port traces to a top-level port that is not declared in the `ClockPeriods` wrapper, you will get an error as shown here:

```
// Error: Unable to trace the iClock
// 'core_rtl_tessent_mbist_c2_controller_inst.BIST_CLK' to a
// valid clock source. Traced the system clock to clock source
// 'clk'. The source of an iClock command must be driven by a
// primary input port of ICL type ClockPort that has been declared
// as a clock source with the add_clocks command.
```

To fix this type of error, you can either add missing `DftControlSettings` to configure the clock multiplexing, or you can add the missing clock port in the `ClockPorts` wrapper.

Examples

Example 1

The following example illustrates a Patterns wrapper that operates the tester and tck at 100ns. The example creates two asynchronous clocks that are needed by the memory BIST controllers operated in TestStep TS0 and TS1.

```
PatternsSpecification(design_name, design_id, pattern_id) {
    Patterns(patterns_name) {
        tester_period      : 100ns ;
        ClockPeriods {
            Clka : 4.5ns ;
            clkb : 5ns ;
        }
        TestStep(TS0) {
            MemoryBist {
                ...
            }
        }
        TestStep(TS1) {
            MemoryBist {
                ...
            }
        }
    }
}
```

Example 2

This example shows how to deal with clock port cloning that may happen during layout. Imagine you had a clock port called clka on your core and after layout this clock port was split into eight ports where the original clock port drive a small percent of the original fanout and the seven new ports clka_1 to clka_7 drives the rest. You need to have this new clock toggling in the test bench whenever clka was toggling. The following example script takes the [PatternsSpecification](#) created by [create_patterns_specification](#) and clones the clka entry in all ClockPeriods wrapper it is already found. The new entries in the ClockPeriods wrapper tells [process_patterns_specification](#) to also start those clocks.

```
set spec [create_patterns_spec]
set new_clka_ports [get_name_list [get_ports clka_*]]
set ClockPeriods_wrappers [get_config_elements ClockPeriods -in $spec -hier]

foreach_in_collection ClockPeriods_wrapper $ClockPeriods_wrappers {
    if {[get_config_value clka -in $ClockPeriods_wrapper -exists]} {
        set period [get_config_value clka -in $ClockPeriods_wrapper]
        foreach new_clka_port $new_clka_ports {
            add_config_element $new_clka_port -in $ClockPeriods_wrapper -value $period
        }
    }
}
```

Related Topics

[create_patterns_specification](#)
[process_patterns_specification](#)

SimulationOptions

Specifies properties that affect the creation of the simulation test benches.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        SimulationOptions {
            testbench_module_name      : format_string ; // default: TB
            monitor_internal_clock_pins : boolean ; // default: on | off
            sdf_info                  : sdf_info_id ;
            emulate_bisr_chains_in_lower_physical_blocks : boolean ;
                                         // default: auto
            logic_bist_debug          : off | bist_registers_and_clock_verify |
                                         setup_and_clock_verify |
                                         setup_and_clock_verify_one_per_ncp;
            LowerPhysicalBlockInstances {
                instance_name : full | graybox | interface ; //repeatable
            }
            TopModules {
                module_name   : file_path ; // repeatable
            }
            SimulationMacros {
                macro_name    : value ;      // repeatable
            }
            NonContactedTestOptions {
                default inout pad decay time : time ;
                PadDecayTimes {
                    port_name   : time ;      // repeatable
                }
            }
        }
    }
}
```

Description

Specifies properties that affect the creation of the simulation test benches.

Arguments

- testbench_module_name : *format_string* ;

A property that specifies the top-level module name in the Verilog test bench file. You can use this property to give a different test bench module name for each test bench so that more than one test bench can be compiled in a single work area of a Verilog simulator at the same time.

The default is TB. The string can also include %d which will be replaced by the name of the current design, or %p which will be replaced by the name of the Patterns wrapper. Using a unique name for the test bench allows you to use a common signal name file to display signals in your simulator including alias name definitions you may have created. The run_testbench_simulations command uses a separate directory to compile each test bench so keeping the testbench_module_name as TB is optimal and recommended.

- monitor_internal_clock_pins : boolean ;

A property that specifies whether the simulation test bench is to monitor the internal clock pins. When specified to on, those pins that have an iClock defined in the PDL are monitored to make sure the right frequency reaches them in simulation. The expected period is computed by the iClock command. The ICL clock pin names are converted to design pin names using the value of the tessent_design_instance property that the tool added to the ICL instances during ICL extraction.

Note

 During layout, ungrouping very often makes internal clock monitoring unusable as the design hierarchy is required to find the internal clock pins. If this happens you must either turn this property off or use a Verilog `define macro to turn off the monitoring, see the “[Clock Monitoring During Simulation](#)” section under the [run_testbench_simulations](#) command for an example of the available macros.

If this property is not specified, it defaults to the value defined in the wrapper `.././SimulationOptions`. This means the PatternsSpecification/SimulationOptions when there is no intermediate PatternsGroup wrapper or PatternsSpecification/PatternsGroup/SimulationOptions otherwise.

- sdf_info : [sdf_info_id](#) ;

A property used to reference a [SdfInfo](#) wrapper. The SDF file identified in the SdfInfo wrapper will be applied to the design when doing the simulation of this pattern set.

- emulate_bisr_chains_in_lower_physical_blocks : on | off | [auto](#) ;

A property that specifies whether the tool should create behavioral code in a side Verilog module (see “[patterns](#)” on page 3774) to emulate the behavior of the BISR segments within the child physical block instances. The valid values are:

[auto](#) — BISR chain emulation is only enabled for those child physical block instances listed in the LowerPhysicalBlockInstances wrapper with a value of “interface” or “graybox”. This is the default.

on — BISR chain emulation is enabled for all child physical block instances. The emulation logic overrides the BISR segments inside the child block instance even if the full view of the child physical block instance has been loaded.

off — BISR chain emulation is disabled for all child physical block instances.

The tool uses hierarchical forces and observe commands to emulate the behavior of the BISR segments on the port of the child physical block instance. This enables you to simulate the BISR operation at the top level if you are using interface or graybox models for the child physical block instances. To use the full simulation model for all instances of a sub physical block module, you can specify the string “`+define+Full_<PhysicalBlockModuleName>`” on the simulator command line; this strips out the behavioral code associated with the block name.

- `logic_bist_debug : off | bist_registers_and_clock_verify | setup_and_clock_verify | setup_and_clock_verify_one_per_ncp ;`

A property that causes the tool to provide extra data to help you debug LogicBIST simulation mismatches. Refer to “[Pattern Mismatch Debugging](#)” in the *Hybrid TK/LBIST User’s Manual* for details. The valid values are:

`off` — The tool does not provide extra data for debugging.

`bist_registers_and_clock_verify` — The simulation generates a mismatch showing when the PRPG, MISR, and low-power registers (if present) diverge from their expected values. This can help reduce debug time because it identifies the scan pattern at which the register(s) begin to diverge.

`setup_and_clock_verify` — The tool creates a LogicBIST pattern that verifies that the controller is setup correctly, and then it applies 256 serial patterns to exercise the full NCP count range. During this test, the tool reduces the shift length to decrease the simulation runtime. Because of the reduced shift length, the MISR signature is not examined at the end of the pattern. If a clock does not generate the expected number of pulses for a given NCP, the tool generates a simulation mismatch.

`setup_and_clock_verify_one_per_ncp` — The same as `setup_and_clock_verify`, except the tool only applies a single scan pattern for each NCP.

- `LowerPhysicalBlockInstances/instance_name : full | graybox | interface ;`

A data wrapper that specifies which view of the lower physical block instances is to be used for this simulation. By default, the interface view (the one containing only the port definition of the instance) is used. If the pattern set operates controllers or instruments found inside some lower physical block instances, the full view of those instances needs to be used. The Patterns wrapper created by the [create_patterns_specification](#) command automatically adds the correct instances that correspond to the controller that it added in the given pattern set. You only need to edit this property if you chose to modify the group of controllers the given patterns wrapper operates on. It is possible to use the graybox value only when a graybox file is available for a lower physical block instance when generated with the “[write_design -tsdb -graybox](#)” command and switch. Selecting this value will generate a simulation testbench for the [ICLNetworkVerify](#) wrapper that automatically excludes all ICL instances that are not used for scan (ICL module attribute [keep_active_during_scan_test](#) is false).

- `TopModules/module_name : file_path ;`

Many modules can be added to the simulation of the parent [Patterns\(pattern_name\)](#) wrapper in which the SimulationOptions/TopModules resides. For each module `module_name`, the file `file_path` in which this module is found is required and only Verilog files are allowed. During the simulation of the Patterns, the file specified by `file_path` is compiled at the same time as the test bench and then, the module `module_name` will be loaded during simulation.

This is equivalent to doing the following on the [run_testbench_simulations](#) command line:

```
run_testbench_simulations -select <pattern_name> \
    -extra_top_modules [list <module_name> ...] \
    -extra_verilog_files [list <file_path> ...] \
    ...
```

Since the TopModules can be specified on a per Patterns basis, it allows for finer control of the simulation process but without having to call the [run_testbench_simulations](#) command for each Patterns(*pattern_name*).

- SimulationMacros/*macro_name* : *value* ;

Each macro *macro_name* and optional value *value* specified in this wrapper translate to the command line argument '+define+...' when the test bench and top modules are compiled during simulation.

For example, for the following wrapper inside a parent [Patterns\(pattern_name\)](#)/SimulationOptions wrapper:

```
SimulationMacros {
    enable_faults;
    fault_type : 2;
}
```

Translate to the following command line arguments when compiling the test bench and top modules:

```
+define+enable_faults
+define+fault_type=2
```

This is equivalent to doing the following on the [run_testbench_simulations](#) command line:

```
run_testbench_simulations -select <pattern_name> \
    -simulation_macro_definitions [list enable_faults fault_type=2] \
    ...
```

Since the SimulationMacros can be specified on a per Patterns basis, it allows for finer control of the simulation process but without having to call the [run_testbench_simulations](#) command for each Patterns(*pattern_name*).

- NonContactedTestOptions/default inout pad decay time : *time* ;

A property that specifies the default inout port decay time that is utilized in [leakage_non_contacted](#) testing. Unique port decay times can be specified in the PadDecayTimes wrapper.

- NonContactedTestOptions/PadDecayTimes/*port_name* : *time* ;

A repeatable property that specifies an inout port name and pad decay time. These attributes are utilized in [leakage_non_contacted](#) testing. If not specified, the port decay time will default to the default inout pad decay time.

Related Topics

[create_patterns_specification](#)

[process_patterns_specification](#)

AdvancedOptions

Specifies special advanced properties that affect various aspects of the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        AdvancedOptions {
            parameter_file           : file_path ;
            Parameters {
                parameter_name : parameter_value ; // repeatable
            }
            include_all_power_pins   : on | off | auto ;
            include_only_used_pins   : on | off | auto ;
            exclude_initial_ireset_from_patterns : on | off ;
            skip_test_setup          : on | off ;
            skip_test_end            : on | off ;
            procfile_name            : file_name ;
            ireset_in_test_setup_proc : on | off ;
            no_initialization       : on | off ;
            // defaults to ../../AdvancedOptions/procfile_name
            ConstantPortSettings {
                // cumulates and overrides constant port settings in
                // ../../AdvancedOptions
                port_name             : 0 | 1 ;
            }
        }
    }
}
```

Description

The AdvancedOptions wrapper is used to specify special advanced properties that affect various aspects of the pattern set.

See the [LoadBoardInfo](#) wrapper description and `load_board_info` property for more control over load board contacted ports and loopbacks.

Arguments

- `parameter_file : file_path ;`
A property that specifies the name of a file containing parameters used to configure the `write_patterns` command. For a description of the available parameters, see the “[Parameter File Format and Keywords](#)” on page 3937. When the property is not specified, the value of the `../../AdvancedOptions/parameter_file` property is used.
- `Parameters/parameter_name : parameter_value ;`
A wrapper that specifies parameter-value pairs used to configure the `write_patterns` command. A parameter specified in this wrapper overrides the same parameter specified in the specified `parameter_file`. For description of the available parameters, see the “[Parameter File Format and Keywords](#)” on page 3937, The list of parameters found in this wrapper are added to any parameter specified in the `../../AdvancedOptions/Parameters` wrapper.

- `include_all_power_pins : on | off | auto ;`

A property that specifies whether or not to include power and ground ports in the patterns file. The power and ground ports are specified using the “`set_attribute_value -name function -value power | ground`” command. When set to auto, the power and ground ports are included in the simulation test benches but excluded from the manufacturing patterns file because the power and ground pins are not driven by the tester channels but directly from the load board.

Depending on its value, this property takes control of the value of the parameter `ALL_EXCLUDE_POWER_GROUND`, which can be set directly in the Parameters wrapper.

- The `ALL_EXCLUDE_POWER_GROUND` parameter keyword is used only when the `include_all_power_pins` property is set to auto (the default) or is not set.
- The `ALL_EXCLUDE_POWER_GROUND` parameter keyword is overridden when the `include_all_power_pins` property is set to on or off.
- `include_only_used_pins : on | off | auto ;`

A property that specifies whether to include all non-power or ground ports in the patterns file or only the one that are actually used in the current pattern set. The used port list is the list of ports explicitly forced or sampled at least once in the pattern set.

Depending on its value, this property takes control of the value of the parameter `ALL_EXCLUDE_UNUSED`, which can be set directly in the Parameters wrapper.

- When set to auto, the property defaults to on when the usage property (see [PatternsSpecification](#)) is `manufacturing_test`; otherwise, it defaults to off.
- The `ALL_EXCLUDE_UNUSED` parameter keyword is used only when the `include_only_used_pins` property is set to auto (the default) or is not set.
- The `ALL_EXCLUDE_UNUSED` parameter keyword is overridden when the `include_only_used_pins` property is set to on or off.

See [LoadBoardInfo](#) wrapper description and `load_board_info` property for more control over load board contacted ports and loopbacks.

- `exclude_initial_ireset_from_patterns : on | off ;`

A property that specifies whether or not to include an asynchronous reset of the IJTAG network in the manufacturing patterns. When set to “off”, the manufacturing patterns will begin with an asynchronous reset of the IJTAG network. When set to “on”, the pattern will begin the execution while assuming that the IJTAG network is in a reset state.

When this property is set to “on”, the IJTAG network must have been restored to its reset state otherwise the execution of the current pattern will fail. In order to bring back the IJTAG network to a reset state, a previous pattern must have been generated with the [TestStep](#)/AdvancedOptions/network_end_state : reset and executed before the current pattern is executed.

This property is typically used when running the memoryBIST pre-repair patterns and memoryBISR fuse programming patterns when implementing the memory repair flow. It is important to disable the initial IJTAG reset in these patterns in order to preserve any memory repair information that was calculated in previous patterns.

Note that this property removes the initial IJTAG reset from the manufacturing patterns only. When generating simulation testbenches, the initial IJTAG reset is applied in order to ensure a valid IJTAG network state during simulation.

- `skip_test_setup : on | off ;`

A property that specifies that the optional `test_setup` procedure that was loaded from the identified procfile using the [Patterns](#)/`profile_name` property is to be skipped from the current patterns file. If you do that, the pattern set is no longer self contained. In manufacturing test, the `test_setup` pattern must be applied to the device once and the state of the device that was obtained by the `test_setup` procedure, and on which the current pattern set depends on, must be maintained by the tester before running the current pattern set. This typically involves maintaining the power to the device and the value of input pins when switching from one pattern set to the next.

- `skip_test_end : on | off ;`

A property that specifies that the optional `test_end` procedure that was loaded from the identified procfile using the [Patterns](#)/`profile_name` property command is to be skipped from the current patterns file

- `procfile_name : file_path ;`

A property that specifies a procfile name that contains a `test_setup` and/or a `test_end` procedure. This property, when unspecified, takes its value from the `../..AdvancedOptions/profile_name` property found in the [PatternsSpecification](#) or PatternsGroup wrapper.

The `file_path` must have one of the format:

- It must be a null string. In this case, no profile is used.
- It must be an absolute path starting with a “/”.
- It must be a relative path starting with “./”. In this case, the file must exist relative to the current working directory.
- It must be a simple file name with no directory path. In this case, the file must exist beside the ICL file.
- `ireset_in_test_setup_proc : on | off ;`

A property which, when set to “on”, will suppress the “iReset” being issued before the first pattern set generated for the Patterns wrapper being processed.

A `test_setup` procedure must be provided in the procfile specified by the “`procfile_name`” property in the same AdvancedOptions wrapper.

It is your responsibility to put appropriate events into the `test_setup` procedure that will reset the IJTAG network.

The effect of the “ireset_in_test_setup_proc” being “on” is to add the “-ireset_in_test_setup_proc” to the “open_pattern_set” command for the first pattern set generated by for the Patterns wrapper. See the [open_pattern_set](#) command for more information.

- no_initialization : on | [off](#) ;

A property that specifies creating the initialization cycle in the pattern file. The default is off. This property, when unspecified, takes its value from the property specified in the following wrapper:

[PatternsSpecification](#)/AdvancedOptions

- ConstantPortSettings/port_name : 0 | 1 ;

A data wrapper that specifies the ports that must be held constant throughout the entire pattern set. This list is typically empty. The ConstantPortSettings wrapper can be used in the [BSDL-only flow](#) (also refer to [BoundaryScan](#) for further information). Input ports that were constrained prior to running [extract_icl](#) are automatically reported in the ICL file and they are automatically forced when [process_patterns_specification](#) is called. Port settings listed in this wrapper accumulate and replace the port settings defined in [..../AdvancedOptions/ConstantPortSettings](#) wrapper found in the [PatternsSpecification](#) and [PatternsGroup](#) wrappers.

If you have used [add_dft_signals](#) scan_en to define top level ports as DFT signals of type scan_en, the [create_patterns_specification](#) command will automatically add the scan enable ports to the AdvancedOptions/ConstantPortSettings wrapper such that they are automatically asserted off during the pattern generation. If you have other scan enable ports not identified with the [add_dft_signals](#), you need to add them yourself in the [create PatternsSpecification](#) using this script:

```
set spec [create_patterns_specification]
# Get a collection object to the
# 'ConstantPortSettings' wrapper
set ConstantPortSettings \
[get_config_value -in $spec -object \
AdvancedOptions/ConstantPortSettings]

# Add entries to 'ConstantPortSettings'
# wrapper for some ports that must be
# forced to '0'.
set scan_en_port_list {se1 se2}
foreach scan_en $scan_en_port_list {
    add_config_element $scan_en \
        -in $ConstantPortSettings -value 0
}
```

Examples

The following example uses the include_only_used_pins property to only include the used ports. The generated stil file will exclude the unused ports. By default, all ports would have been listed in the patterns file even if they had no activity.

```
PatternsSpecification(my_design, final, manufacturing) {
    usage : manufacturing_test ;
    manufacturing_patterns_format : stil ;
    Patterns(mbist1) {
        AdvancedOptions {
            include_only_used_pins : on ;
        }
    }
}
```

Related Topics

[create_patterns_specification](#)

[process_patterns_specification](#)

TestStep

Inserts a test step in the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        TestStep(step_name) {
            AdvancedOptions {
                force_voltage(pin_name) : voltage ; // repeatable
                split_patterns_file : on | off ;
                network_end_state : keep | reset | initial ;
                IjtagRetargetingOptions {
                    merge_irunloop_only : on | off ;
                }
            }
            BoundaryScan {
            }
            MemoryBist {
            }
            MemoryBisr {
            }
            LogicBist {
            }
        }
    }
}
```

Description

The TestStep wrapper is used to insert a test step in the pattern set. The test step includes the selection of Boundary Scan tests, MemoryBIST or LogicBIST tests.

Arguments

- *step_name*
A string that uniquely identifies the step within a pattern set. All procedure_step_name and test_step_name strings must be unique within a Patterns wrapper. The string is only allowed to contains letters, numbers, and underscores.
- AdvancedOptions/force_voltage(*pin_name*) : *voltage* ;
A repeatable property that specifies that the current test step needs a special voltage to be applied on a specific port of the device. This is typically used in TestStep where MemoryBisr is used to program the fusebox. The property does not actually force the voltage. It only generates a special annotation near the top of the patterns file to let the test engineer know that test program commands are needed to supply the needed voltage on the given port. As soon as a single force_voltage property is specified, the split_patterns_file : on; property is implied to allow the insertion of the test program commands needed to apply the voltage values.

Below is an example of the special annotation inside a stil file:

```
Ann { * Before applying this pattern, the following actions are required: * }
Ann { * - Apply +0.25V on port PA * }
Ann { * - Apply -0.25V on port PB * }
```

- AdvancedOptions/split_patterns_file : on | off ;

A property that causes the pattern set to be split into multiple files. The file containing the previous patterns is closed and a new one is created for the current step. The file name is the name of the Patterns wrapper initially. When a step with split_patterns_file set to on is reached, the new file name consists of the name of the Patterns wrapper concatenated with an underscore to the name of the TestStep wrapper as shown here:

```
<pattern_name>_<test_step_name>
```

The split patterns file contains a special annotation near the top of the file that reads like this:

```
Ann { * This pattern file requires some actions taken on the ATE. * }
Ann { * This pattern assumes it is following another one: * }
Ann { * the power and clocks must be kept alive between * }
Ann { * the two patterns and all pins kept their state. * }
```

- AdvancedOptions/network_end_state : keep | reset | initial ;

A property that specifies the action to take at the end of the TestStep. The options for this property are named the same and perform the identical functions as the close_pattern_set command's arguments. For details, see the [close_pattern_set](#) command.

- IjtagRetargetingOptions/merge_irunloop_only : on | off ;

A property that specifies if iApply statements of concurrent iCalls are allowed to be merged during the processing of an iMerge block. This property and its options perform the identical function as “set_ijtag_retargeting_options [-merge_irunloop_only](#)”. Refer to the [set_ijtag_retargeting_options](#) command for more information.

Examples

The following example defines two TestStep wrappers in series. The first one runs memory BIST controllers while the second ones runs a MemoryBisr controller.

```
PatternsSpecification(mychip, gate, signoff) {
    Patterns(P1) {
        TestStep(TS1) {
            MemoryBist {
                Controller(corea.ctrl1) {
                }
                Controller(corea.ctrl2) {
                }
            }
        }
        TestStep(TS2) {
            MemoryBisr {
                Controller(ctrl1) {
                }
            }
        }
    }
}
```

Related Topics

[create_patterns_specification](#)

[process_patterns_specification](#)

BoundaryScan

Specifies the boundary scan tests to be generated.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        TestStep(step_name) {
            BoundaryScan {
                bsdl_file : file_path ;
                bsdl_package_directories : dir_path | auto ;
                enable_group_info : enable_group_info_id, ... ;
                bonding_configuration : name ;
                pin_action_verbosity : on | off ;
                active_control_cells : unlimited | integer ;
                RunTest(test) {
                }
            }
        }
    }
}
```

Description

The BoundaryScan wrapper specifies the boundary scan tests to be generated.

As illustrated in its syntax, the BoundaryScan specification consists of RunTest wrappers and properties that will affect all RunTest wrappers.

The repeatable RunTest wrapper defines the tests to be generated in the current pattern set. At least one RunTest wrapper is required, and no more than one BoundaryScan wrapper may be specified per TestStep wrapper.

Arguments

- bsdl_file : *file_path* ;

A property that specifies the path and filename of the Boundary Scan Description Language (BSDL) file to be used for BSDL-only flow pattern generation. In the BSDL-only pattern generation flow, only the BSDL file is required. You do not need to specify the BSDL file if it can be found in an instrument container in the TSDB directory. Packages can optionally be specified in the BSDL file and included with the bsdl_package_directories argument. For more information, refer to “[BSDL-Only Flow](#)” in the *Tessent BoundaryScan User’s Manual*.

Note

 In the BSDL-only flow, it may not be possible to use the [run_testbench_simulations](#) command since the data needed to run this command may not be available.

- `bsdl_package_directories : dir_path | auto ;`

A property that specifies a single path, or a list of paths to the package include directories. If `dir_path` is not specified, or if `auto` is specified, the path is assumed to be the same directory where the BSDL file is located.

- `enable_group_info : enable_group_info_id, ... ;`

A property that specifies one or more `EnableGroupInfo` wrappers to use for the boundary scan patterns. If more than one `EnableGroupInfo` wrapper is specified, the tool automatically extends the test to multiple sequential tests that enable each `EnableGroupInfo` wrapper separately.

All enable cells not part of the listed enable groups, whether they are part of a not listed enable group or part of no enable group, are added to a new artificial group and tested all simultaneously.

If an `EnableGroupInfo(enable_group_info_id)` wrapper is not found in the `PatternsSpecification` wrapper, the tool generates an error listing the `enable_group_info_ids` that are available. If the user specifies “`enable_group_info : *;`”, the tool automatically adds all available `enable_group_info_ids` to this property.

- `bonding_configuration : name ;`

A property that selects which bonding configuration the current patterns will be generated for. You must specify this property if your Tessent BoundaryScan hardware was generated with one or more `BondingConfigurations`. The name string must refer to a valid `BondingConfigurations(name)`. Tessent Shell will search for the corresponding BSDL file `<design_name>_<name>.bsdl` in the TSDB.

This property cannot be used if `bsdl_file` is specified.

- `pin_action_verbosity : on | off ;`

A property that enables you to specify whether your simulator displays comments each time a pin is compared to a specific value during test bench simulation. Be aware that adding comments for each output comparison increases pattern storage size.

- `active_control_cells : unlimited | integer ;`

A property that specifies how many enable boundary-scan cells can be active simultaneously during the clamp and output tests. Valid integer values are 1 or greater.

A large value generates a shorter overall test. Using a small value reduces the noise of simultaneously switching outputs created by test fixtures and package inductance.

If the `enable_group_info` property is specified and the integer is smaller than the number of control cells in an `enable_group`, the `enable group` will be split up and run sequentially in smaller groups of integer cells simultaneously.

Examples

The following example shows the PatternsSpecification of a standard set of 1149.1 tests with a reduction in the patterns storage size due to turning off pin action verbosity and a reduction in switching activity due to limiting the number of simultaneously active control cells.

```
PatternsSpecification(CHIP,signoff) {
    Patterns(std_test) {
        TestStep(TS1) {
            BoundaryScan {
                pin_action_verbosity: off ;
                active_control_cells: 1 ;
                RunTest(test_logic_reset){}
                RunTest(inst_reg){}
                RunTest(id_Reg){}
                RunTest(bypass_reg){}
                RunTest(bscan_reg){}
                RunTest(input){}
                RunTest(sample){}
                RunTest(highz){}
                RunTest(output_clamp){}
            }
        }
    }
}
```

RunTest

Defines a TAP or IO test to be run.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        TestStep {
            BoundaryScan {
                RunTest(test_name) {
                    AdvancedOptions {
                        checker_board : on | off ;
                    }
                    NonContactedPinOptions {
                        LeakageOptions {
                            logic_level : both | iih | iil ;
                            number_of_cycles : int ; default: 0
                        }
                        TriStateEnableOptions {
                            test_path : both | local | global ;
                            logic_level : both | high | low ;
                            number_of_rti_cycles : integer ; // default: 1
                        }
                    }
                    CellCompares {
                        cell_number : 0 | 1 | x ; // repeatable
                    }
                    CellSettings {
                        cell_number : 0 | 1 ; // repeatable
                    }
                    MaskedPins {
                        pin_name : normal | masked ; // repeatable
                    }
                    PinSettingsOverride {
                        pin_name : 0 | 1 | z ; // repeatable
                    }
                    PinComparesOverride {
                        pin_name : 0 | 1 | x ; // repeatable
                    }
                }
            }
        }
    }
}
```

Description

The RunTest wrapper defines a TAP or IO test to be run. Using the sub-wrappers in the RunTest wrapper, you can customize pin or cell options that will apply to this test only. This wrapper needs to be repeated once for every test type you want to run.

Arguments

- *test_name*

A string that specifies the type of test to run.

The valid values for *test_name* and a description of each test type follow:

Value	Description
bscan_reg	Ensures the boundary scan data register responds properly to the TAP state machine, and that the boundary scan register can be loaded. This algorithm uses the SAMPLE instruction, shifting data through the boundary scan chain only. Testing whether the boundary scan chain can actually capture and update data to the pins is performed as part of the input and output algorithms, and is therefore not included as part of the bscan_reg algorithm.
bypass_reg	Ensures that the data register state machine is functioning properly and that the bypass register can be loaded. This algorithm scans through the bypass register each time, comparing the bypass register capture value to 0. The algorithm also steps through all possible state transitions associated with a data register scan operation. If the TAP does not track states of the FSM (finite state machine) properly, the 11110 pattern will not appear on TDO in subsequent data register scan operations. The algorithm performs a final check of the bypass operation codes by scanning a 11111 pattern through the bypass register using every instruction opcode documented in the BSDL file as selecting the Bypass register.
clamp	This test verifies the CLAMP instruction. It consists of creating various clamp states and verifying that the output and the bidirectional pins are in the correct states. The clamp test exercises each boundary-scan cell with each pin driven in the high and the low state. In addition, the clamp test performs a scan-through test of the bypass register. The execution is conditional on the existence of the TAP CLAMP instruction code.
controlr	This test verifies that all controlr cells are reset to their disable state after a TAP controller reset. Execution is conditional upon the presence of at least one controlr cell.
disabled_outputs	This test disables all pads in the EXTEST mode by scanning a disabling value in all their enable cells. Tests the output for high impedance. Execution is conditional on the presence of at least one control cell or one open drain/source pad.
dot6_ac_00	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.

Value	Description
dot6_ac_01	<p>This test measures AC parameters V_{hyst_edge} and T_{hyst} on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.</p> <p>This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.</p>
dot6_ac_10	<p>This test measures AC parameters V_{hyst_edge} and T_{hyst} on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.</p> <p>This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.</p>
dot6_ac_11	<p>This test measures AC parameters V_{hyst_edge} and T_{hyst} on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.</p>
dot6_ac_input	<p>This test verifies the AC operation of the 1149.6 receivers. It performs the following tasks:</p> <ul style="list-style-type: none"> Applies a checkerboard pattern to contacted AC input/bidirectional pins. Disables all output/bidirectional pads. Applies a valid AC waveform to contacted input and inout pins. Tests the “no transition” detection capability, if present, by applying two consecutive test patterns while keeping the AC pin level constant. If the pad contains either a Low-Pass filter or an embedded High-Pass filter, the absence of transition in the second pattern will make its corresponding boundary-scan cell capture its default value. Uses both EXTEST_PULSE and EXTEST_TRAIN codes.
dot6_ac_output	<p>This test verifies the AC operation of the 1149.6 transmitters and loopbacks. It performs the following tasks:</p> <ul style="list-style-type: none"> Enables all AC output pads. Ignores and turns off all non-AC pads, if possible. Uses both AC and DC loopbacks if present. Uses both EXTEST_PULSE and EXTEST_TRAIN codes. Alternatively disables and enables ACSelect cells. Runs the test on one Enable Group at a time. Strobes AC waveforms on all AC contacted outputs.

Value	Description
dot6_ac_select_cells	<p>This test verifies that the netlist properly implements the AC grouping described in the BSDL. Only one ACSelect cell is enabled at a time, and its action is verified by checking that all AC output pins in its fanout properly toggle during the RunTestIdle state when the EXTEST_PULSE instruction is loaded.</p> <p>This test is necessary only during netlist verification in the design flow. In manufacturing, the dot6_ac_output test ensures that all AC select cells are properly operating by enabling or disabling them all at once.</p>
dot6_dc_00	<p>This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test.</p>
dot6_dc_01	<p>This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test.</p> <p>This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.</p>
dot6_dc_10	<p>This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test.</p> <p>This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.</p>
dot6_dc_11	<p>This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test.</p>
dot6_dc_input	<p>This 1149.6 test verifies the behavior of all 1149.6 AC input pins when operating in EXTEST mode. Just as the non-AC (in other words, DC) pins “input” test, this pattern applies zero and one logic values at the AC pins and checks that their 1149.6 test-receiver’s associated bscan cells capture the same value. In addition, whenever the target tester supports it, the test will also verify that all AC pins can reliably detect invalid input voltage levels sitting in between the “zero” and “one” threshold voltages.</p>

Value	Description
dot6_dc_output	This 1149.6 test is equivalent to the existing output IO test for normal DC pads. Just as in the output test, the dot6_dc_output test applies checkerboard values, activates output cells one enable group at a time, and uses load board loopbacks as well as IO internal loopbacks. In addition, this test also covers 1149.6-specific features such as coupling capacitors discharge time, ACSelect cells, and output inversion during the RunTestIdle state of the TAP.
highz	This test verifies the operation of the HIGHZ instruction code. The pattern loads the HIGHZ code into the TAP, driving all output and bidirectional pins into high-impedance states. Note that this test actually executes the following three tests: disabled_outputs, highz_mode, and controlr.
highz_mode	<p>This test disables all pads via the TAP HIGHZ instruction code. It tests all outputs for high impedance.</p> <p>The execution is conditional to the existence of a TAP HIGHZ instruction code.</p>
id_reg	Ensures that the ID register is responding properly to the TAP data register state machine. This algorithm compares the capture values of the device ID and status registers whenever a data register scan operation is performed. The algorithm steps through all possible state transitions associated with a data register scan operation. The execution is conditional to the existence of a TAP IDCODE instruction code.
iih	Presets input pins so that the tester can drive the pins to their high values and measure the current of these pins. The tester stops the test to measure current, then resumes the test.
iil	Presets input pins so that the tester can drive the pins to their low values and measure the current of these pins. The tester stops the test to measure current, then resumes the test.
input	<p>This test verifies that the boundary-scan cells perform a proper capture of input data using standard IEEE-1149.1 timing. It tests all input, observe-only, and clock boundary-scan cells when the TAP instruction register is loaded with the EXTEST instruction code. In addition, it also tests the input direction of all bidirectional cells. The test turns off all pad drivers, including the bidirectional ones, and applies alternate EVEN and ODD checkerboard patterns to all input and bidirectional contacted pins that have tester control capabilities. Cells are pre-loaded with their pin state's opposite value. All non-X cell captured values are compared at TDO including those resulting from the following:</p> <p>Cells capturing themselves or a constant value in EXTEST mode.</p> <p>When a pin is tied to a constant value by the load board.</p> <p>The presence of an internal or external pull-up or pull-down resistor at the output of an uncontrollable bidirectional pin.</p>

Value	Description
inst_reg	<p>This test verifies that the instruction register can be loaded. The two least significant bits are compared to “01” on every instruction register scan operation. This algorithm steps through all possible state transitions associated with an instruction register scan operation.</p> <p>If the TAP does not track states of the FSM (finite state machine) properly, the standard 01 bits of the instruction register do not appear in a subsequent instruction register operation.</p>
leakage_non_contacted	<p>This test will drive a value to an inout port and observe the value after a specified number of cycles.</p> <p>The leakage_non_contact test can only be used if there are non-contacted inout ports. These non-contacted inout ports can be described in the PatternsSpecification/LoadBoardInfo/TesterInterface wrapper with port properties of control:none and observation:none set.</p> <p> Note: The Verilog test bench will pass only if you specify the pad decay time in the SimulationOptions/NonContactedTestOptions/PadDecayTimes wrapper with a time greater than that specified in the NonContactedPinOptions/LeakageOptions/number_of_cycles property.</p>
tristate_enable_non_contacted	<p>This test verifies if the pad driver of a bidirectional non-contacted pad can be disabled by the force_disable signal and the local enable signal. The RunTest/NonContactedPinOptions/TriStateEnableOptions wrapper defines the types of tests to run (global or local) and their parameters. This wrapper can only be used with the tristate_enable_non_contacted test.</p> <p>The pad decay times should be defined in the SimulationOptions/NonContactedTestOptions/PadDecayTimes wrapper for the testbench.</p>

Value	Description
output	<p>This test verifies that the boundary-scan cells perform a proper capture of output data using standard IEEE 1149.1 timing. It also checks the preload functionality of the SAMPLE/ PRELOAD instruction. In addition, it tests output and bidirectional boundary-scan cells when the TAP instruction register is loaded with the EXTEST instruction. Output and bidirectional pad drivers are turned on sequentially on a per group basis so as not to exceed the values specified with the active_control_cells property. All enabled output and bidirectional cells are loaded alternatively with EVEN and ODD checkerboard patterns. The output and bidirectional pins are strobed if they are both contacted and observable. All non-X values captured by the boundary-scan cells are compared at TDO, including the ones from the following:</p> <ul style="list-style-type: none"> Cells capturing themselves or a constant value in EXTEST mode. Enabled bidirectional boundary-scan cells that capture the pin state in EXTEST mode. Disabled bidirectional boundary-scan cells that have an internal or external pull-up/pull-down resistor. Input pins driven by a load board loopback from an output or bidirectional pin. Input cell captured value for all enabled bidirectional pads controlled by the combination of one output and one input or sample boundary-scan cell.
output_clamp	<p>This test runs the output test with an added short clamp test. During the output test, after each load of the boundary-scan register and each strobing of the enabled output and input pins, the following is performed:</p> <ul style="list-style-type: none"> The TAP instruction register is reprogrammed with the CLAMP instruction code. The test verifies whether the BYPASS register is selected. The output pins are strobed again to verify that they are to retain the initial output state value in CLAMP mode. The TAP instruction register is then reprogrammed with the EXTEST instruction, and the output test resumes from there.
sample	<p>This test verifies that the boundary-scan cells perform a proper capture of input data, like the input test, with the exception of the bidirectional pins. This algorithm uses the SAMPLE instruction code instead of the EXTEST instruction code which means the pad drivers of bidirectional pins cannot be turned off.</p>
test_logic_reset	<p>This test verifies that the TAP is being reset properly. Note that the boundary scan chain contains safe data values, and that either the EXTEST or SAMPLE instruction is active.</p>

Value	Description
voh	Presets output pins so that the tester can drive the pins to their high values and measure the voltage of these pins. The tester stops the test to measure voltage, then resumes the test.
vol	Presets output pins so that the tester can drive the pins to their low values and measure the voltage of these pins. The tester stops the test to measure voltage, then resumes the test.

- AdvancedOptions/checker_board : on | off ;

A property that enables or disables applying checkerboard vectors. Valid settings are as follows:

on	Enables checkerboard vectors which alternate logic zeroes and ones from one pin to the next, following the order of the boundary-scan register. This increases the chances of detecting short-circuits between two adjacent pins. Default IO test vectors are checkerboard.
off	Disables checkerboard, therefore applying boundary-scan load vectors made of all zeroes or all ones. This leads to easier debugging and simplifies usability.

This property affects the following tests: input, output, clamp, sample, dot6_ac_input, dot6_dc_input, dot6_ac_output, dot6_dc_output, and dot6_ac_select_cells.

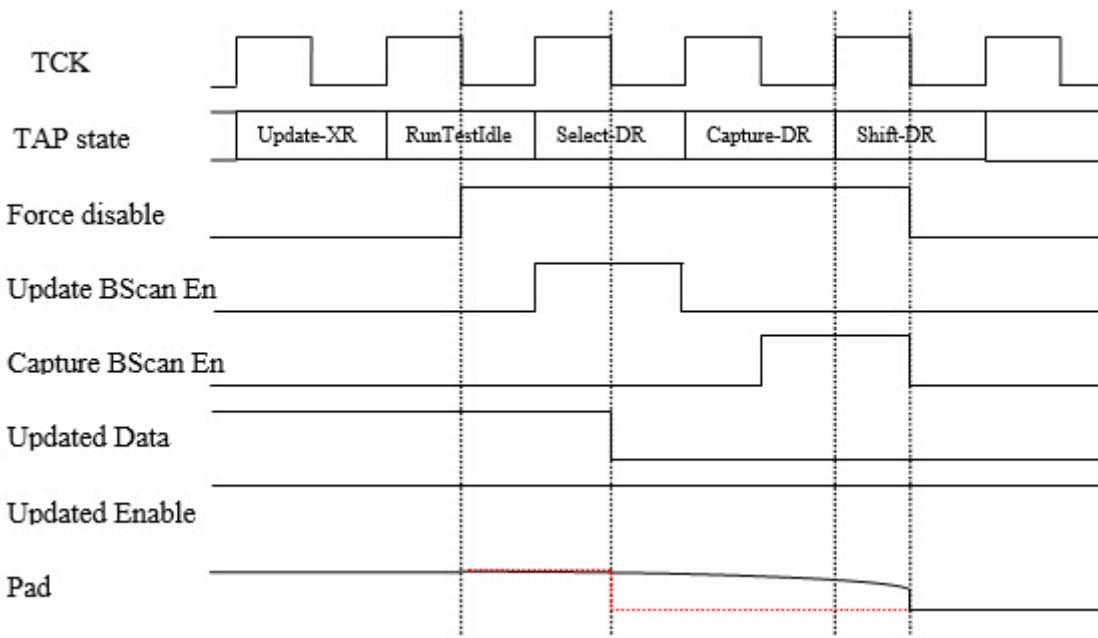
If the value is set to off, individual boundary-scan load vectors are still named EVEN and ODD in the output test bench. However, EVEN will mean logic 0 for all IO pins and ODD will mean logic 1 for all IO pins.

- NonContactedPinOptions/LeakageOptions/logic_level : both | iih | iil ;
A property that specifies the logic levels to be tested with the leakage_non_contacted test. This property can only be used with the leakage_non_contacted test. Specifying iih tests the leakage current with a value of 1 driven on the inout port. Specifying iil tests the leakage current with a value of 0 driven on the inout port. The default value of “both” with test the leakage current with both a 0 and 1 value driven.
- NonContactedPinOptions/LeakageOptions/number_of_cycles : *int* ;
A property that specifies the number of wait cycles between driving the test value and checking the value on a non-contacted inout port. This property can only be used with the leakage_non_contacted test. The default value is “0”. The specified wait time should be smaller than the pad decay time specified for the ports that are tested. The pad decay time is specified in the PatternsSpecification/Patterns/[SimulationOptions](#)/NonContactedTestOptions/PadDecayTimes wrapper.
- NonContactedPinOptions/TriStateEnableOptions/test_path : both | local | global ;
A property that specifies whether the global or the local test is created. By default, both tests are created.

The global test charges the pads with the value specified in the TriStateEnableOptions/logic_level property, and loads the data bscan cell with the opposite value. The update element is not strobed, so the value is not applied to the pad, as shown in [Figure 10-57](#) below. The force_disable signal is asserted, tri-stating the pad, and remains active until the pad state is captured. The updated data, set to the opposite state, is loaded in the SelectDR TAP state, which is done 1.5 cycles before the capture occurs. If the force_disable signal is not able to disable the pad driver because of a faulty circuit, the pad will be driven with the opposite value and that value will be captured, as shown with the red dotted line.

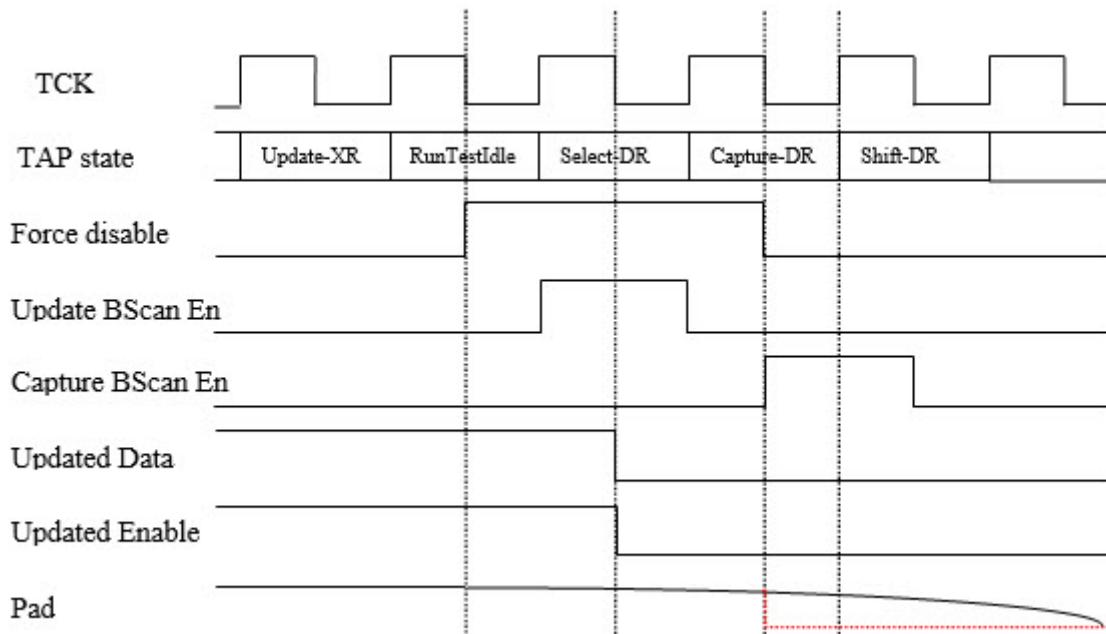
The timing in [Figure 10-57](#) reflects the default setting of “1” for TriStateEnableOptions/number_of_rti_cycles. By using larger settings, the number of cycles between disabling the pad driver and capturing the value can increased. This can be used to test the amount of time the pad can hold a charged value. The pad decay time, or the time a pad can hold a value, is defined in the PatternsSpecification/Patterns/[SimulationOptions](#)/NonContactedTestOptions/PadDecayTimes wrapper.

Figure 10-57. Timing Diagram of the Global tristate_enable_non_contacted Test



The local test charges the pads with the value specified in the TriStateEnableOptions/logic_level property, and loads the data bscan cell with the opposite value. Additionally, the enable cell is loaded with the pad disable value. The update element is not strobed, so the value is not applied to the pad, as shown in [Figure 10-58](#) below. The force_disable signal is asserted, tri-stating the pad as is done with the global test. The updated data (opposite) and enable (disable value), is loaded in the SelectDR TAP state, which is done 1.5 cycles before the capture occurs. Force disable is released in the CaptureDR TAP state, leaving only the enable cell, loaded with the pad disable value, to tri-state the pad. If the enable cell is not able to disable the pad driver because of a faulty circuit, the pad will be driven with the opposite value and that value will be captured, as shown with the red dotted line in the figure.

Figure 10-58. Timing Diagram of the Local tristate_enable_non_contacted Test



- NonContactedPinOptions/TriStateEnableOptions/logic_level : both | high | low ;
A property that specifies the logic level that is used to pre-charge the pads for local and global tristate_enable_non_contacted tests. By default, tests are created for both high and low levels.
- NonContactedPinOptions/TriStateEnableOptions/number_of_rti_cycles : *integer* ;
A property that specifies the number of cycles the tristate_enable_non_contacted global test will remain in the RunTestIdle TAP state. This determines how long the pad will remain undriven before the value is captured 1.5 cycles later. The default value is 1, or one cycle.
- CellCompares/cell_number : 0 | 1 | X ;
A property that enables you to override one or more boundary-scan cells' expected captured values for all scan operations performed during the test. The cell_number is a valid boundary-scan cell number as listed in the BOUNDARY_REGISTER Attribute section of the BSDL file of the design. A setting of X instructs Tessent Shell to ignore the captured and shifted out value of this boundary-scan cell, and a setting of 0 or 1 will instruct it to compare the captured and shifted out value of this boundary-scan cell to this logical value.
- CellSettings/cell_number : 0 | 1 ;
A property that enables you to override the value to be shifted into one or more specific boundary-scan cells for all scan operations performed during the test. The cell_number is a valid boundary-scan cell number as listed in the BOUNDARY_REGISTER Attribute section of the BSDL file of the design. A setting of 0 instructs Tessent Shell to shift in and update the logic 0 value in the specified boundary-scan cell, and a setting of 1 instructs Tessent Shell to shift in and update the logic 1 value in the specified boundary-scan cell.

For the specified cells, the CellSettings wrapper overrides the value for all shift-in and update operations that will be performed during execution of the test specified by `test_name` of the RunTest wrapper. Comparisons of captured pin values are not adjusted accordingly and will have to be comprehended with the appropriate CellCompares and PinComparesOverride settings. Typically, this wrapper is used to force a static value on an output or bidirectional pin of the DUT.

- `MaskedPins/pin_name : masked | normal ;`

A property that specifies that one or more pins will be masked during the IO tests. Any failure related to these pins or to associated boundary-scan cells is not reported during execution of the test. The `pin_name` is a valid port name on the current design.

- `PinSettingsOverride/pin_name : 0 | 1 | Z ;`

A property that specifies a constant value for one or more input or bidirectional pins. The `pin_name` is a valid name of a top-level port (primary input or bidirectional pin). A value of 0, 1, or Z indicates that a logic 0, logic 1, or Z value, respectively, is forced on the pin. This property setting overrides the pin value that the test normally applies.

- `PinComparesOverride/pin_name : 0 | 1 | X ;`

A property that specifies for Tessent Shell to compare one or more output or bidirectional pins to a given value. The `pin_name` is a valid name of a top-level port (primary output or bidirectional pin). A value of 0 or 1 indicates that a logic 0 or logic 1 value, respectively, is compared on the primary pin. A value of X indicates that no value is compared on the primary pin. This property setting overrides the compare value that the test normally applies.

It is recommended to use this wrapper to list all output2 pins that cannot be tri-stated to a compare value of X for highz, highz_mode, disabled_outputs and controlr tests.

Examples

The following example shows a configuration that tests the TAP controller hosting the boundary scan, and a separate set of patterns to test the boundary scan:

```
PatternsSpecification(CHIP,signoff) {
    Patterns(TAP_test) {
        TestStep(TS1) {
            BoundaryScan {
                RunTest(test_logic_reset){}
                RunTest(inst_reg){}
                RunTest(id_Reg){}
                RunTest(bypass_reg){}
            }
        }
    }
    Patterns(bscan_test) {
        TestStep(TS1) {
            BoundaryScan {
                RunTest(bscan_reg){}
                RunTest(input) {
                    AdvancedOptions {
                        checker_board: off ;
                    }
                }
                RunTest(input) {
                    AdvancedOptions {
                        checker_board: on ;
                    }
                }
                RunTest(output_clamp){}
                RunTest(sample){}
                RunTest(highz) {
                    PinComparesOverride {
                        OUT5 : X ;
                        OUT6 : X ;
                    }
                }
            }
        }
    }
}
```

Related Topics

[TestStep](#)

LogicBist

Specifies the LogicBIST tests to be generated.

Usage

```
PatternsSpecification(design_name,usage) {
    Patterns(patterns_name) {
        TestStep(test_step_name) {
            LogicBist {
                CoreInstance(icl_instance_name) {
                    run_mode : hw_default | run_time_prog | burn_in ;
                    burn_in_time : tvalue ;
                    mode_name : mode_name ;
                    misr_comparers : int ;
                    begin_pattern : int | last_pattern ;
                    end_pattern : int | last pattern ;
                    warmup_pattern_count : int ;
                    shift_clock_select : shift clock src | ltest_clock | tck ;
                    DiagnosisOptions {
                        extract_flop_data : on | off ;
                    }
                }
            }
        }
    }
}
```

Description

The LogicBIST wrapper specifies the LogicBIST tests to be generated.

As shown in the syntax above, the LogicBIST specification consists of one or more CoreInstance wrappers that contain the properties that determine how the Hybrid TK/LBIST core is to be run.

Arguments

- **CoreInstance(*icl_instance_name*)**
A required property that specifies the ICL instance to the core containing the Hybrid TK/LBIST controller. If the controller is in the root module, then you specify “.”.
- **run_mode : hw_default | run_time_prog | burn_in ;**
A property that specifies whether the Hybrid TK/LBIST controller runs with the default settings or with scanned-in settings. When set to run_time_prog, the default, the controller is scanned through the IJTAG network to set up the controller according to the properties specified in the CoreInstance wrapper. When set to hw_default, the controller uses hard-coded values that were determined during the DFT process. When set to burn_in, the controller runs continuously for the amount of time specified by the burn_in_time property.

- `burn_in_time : tvalue ;`

An optional switch and time-delay value pair that specifies how long the LogicBIST wafer-level burn-in test should run. You can specify the time values using the following units. The default is 1s.

- s - seconds
- ms - milliseconds
- us - microseconds
- ns - nanoseconds
- ps - picoseconds

- `mode_name : mode_name ;`

A property that selects which LogicBIST mode should be used from the [logic_test_cores](#) directory within the TSDB. During fault simulation, you can change the mode by using the `set_current_mode` command. If the TSDB only contains one mode, the tool will use that mode. If the TSDB contains more than one mode, you must specify which mode to use.

- `mistr_comparisons : int ;`

A property that enables you to specify the number of times the MISR is compared during the LogicBIST run. The number of MISR compares cannot exceed the number of patterns being run, as specified by the `begin_pattern` and `end_pattern` properties. If `DiagnosisOptions/extract_flop_data` is on, the `mistr_comparisons` property specifies the number of times the contents of the scan chains will be scanned. The default is 1.

- `begin_pattern : int | last_pattern ;`

A property that specifies the first LogicBIST pattern to apply. The default is 0.

- `end_pattern : int | last_pattern ;`

A property that specifies the last LogicBIST pattern to apply. The default is `last_pattern`.

- `warmup_pattern_count : int ;`

A property that specifies the first N patterns of a LogicBIST pattern to be masked in order to allow the voltage in the chip to stabilize. When a warm-up value is specified, the PRPG will be seeded with pattern `begin_pattern - N` and the MISR will hold the `begin_pattern` seed until the warm-up patterns are completed. The default is 0.

- `shift_clock_select: shift_clock_src | ltest_clock | tck ;`

A property that specifies the clock input for the Hybrid TK/LBIST controller.

- `DiagnosisOptions/extract_flop_data : on | off ;`

A property that enables the diagnostic mode in which the state of the circuit is frozen after the application of the last pattern and is then the contents of each flip-flop that is serially scanned out via the IJTAG network. The default is off.

Examples

Example 1

The following example shows the PatternsSpecification for a top-level Hybrid TK/LBIST controller in the module CHIP. The single pattern consists of two TestSteps that require the clocks clk100, clk150, and clk200 to be active at the periods shown. The first TestStep runs a total of 8 patterns, beginning with pattern 0. After the last pattern (7) runs, the MISR signature is scanned out and compared. The second TestStep runs only pattern 7, but after the pattern is complete the flip-flop data is scanned out and compared.

```
PatternsSpecification (CHIP,signoff) {
    Patterns (LogicBist) {
        ClockPeriods {
            clk100 : 10ns ;
            clk150 : 6.66667ns ;
            clk200 : 5ns ;
        }
        TestStep (serial_load) {
            LogicBist {
                CoreInstance(CHIP) {
                    run_mode      : run_time_prog ;
                    begin_pattern : 0 ;
                    end_pattern   : 7 ;
                }
            }
        }
        TestStep (diagnostic) {
            LogicBist {
                CoreInstance(CHIP) {
                    run_mode      : run_time_prog ;
                    begin_pattern : 7 ;
                    end_pattern   : 7 ;
                    DiagnosisOptions {
                        extract_flop_data : on ;
                    }
                }
            }
        }
    }
}
```

Example 2

The following example runs the Hybrid TK/LBIST controller from pattern 512 to 524. With 3 misr_comparisons the pattern will be broken up into 3 runs, where the first runs patterns 512 to 515, the second runs patterns 516 to 519, and the last runs patterns 520 to 524. In addition, because warmup_pattern_count is set to 8, the 3 misr_comparisons runs are effectively running patterns 504 to 515, 508 to 519, and 512 to 524, but those 8 warm-up patterns will not contribute to the calculated MISR signatures.

```
PatternsSpecification(top,gate,signoff) {
    Patterns(LogicBist_m8051_1) {
        ClockPeriods {
            refclk : 10.00ns ;
        }
        TestStep(serial_load) {
            LogicBist {
                CoreInstance(m8051_wrap_inst_m8051_inst1) {
                    run_mode : run_time_prog ;
                    begin_pattern : 512 ;
                    end_pattern : 524 ;
                    misr_comparisons : 3 ;
                    warmup_pattern_count : 8 ;
                }
            }
        }
    }
}
```

Example 3

The following example runs the Hybrid TK/LBIST controller in wafer-level burn-in mode for 24 hours (86400 seconds).

```
PatternsSpecification(top,gate,signoff) {
    Patterns(LogicBist_m8051_1) {
        ClockPeriods {
            refclk : 10.00ns ;
        }
        TestStep(serial_load) {
            LogicBist {
                CoreInstance(m8051_wrap_inst_m8051_inst1) {
                    run_mode : burn_in ;
                    burn_in_time : 86.4e+3s;
                }
            }
        }
    }
}
```

MemoryBisr

Controls BISR circuitry in your design.

Usage

```
PatternsSpecification(design_name,design_id, pattern_id) {  
    Patterns(pattern_name) {  
        TestStep(name) {  
            MemoryBisr {  
                run_mode : bisr_chain_access | autonomous | fuse_box_access ;  
                Controller() {  
                }  
            }  
        }  
    }  
}
```

Description

The MemoryBisr wrapper is used for controlling BISR circuitry in your design.

At the block or sub-block design level, the [Controller](#) wrapper and run_mode property are used to verify the operation of BISR chains even though the chains may not yet be connected to a BISR controller.

At the chip design level, all parameters are used to verify the operation of BISR controller itself and its connections to the BISR chains and fuse box.

Arguments

- run_mode : bisr_chain_access | autonomous | fuse_box_access ;

A property used for BISR verification at the chip top, block, or sub-block level specifying the mode of operation of the BISR circuitry. The property must be specified because there is no default value for it. At the block and sub-block level, only the bisr_chain_access mode is valid. All modes are valid at the chip top level when the chains are connected to a BISR controller. The following table describes the three available modes.

bisr_chain_access	<p>Enables access to BISR chains. At the block/sub-block level, BISR chains are typically directly connected to top-level ports. At the chip top level, the BISR controller enables access to the BISR chains using the TDI and TDO top-level TAP ports.</p> <p>Several options are available to transfer repair information from BIRA modules to BISR registers, and serially load/unload user-defined values in selected BISR chains. These options are explained in the BisrChainAccessOptions wrapper description.</p>
-------------------	--

<u>autonomous</u>	In this mode, the controller operation does not require any data patterns from the tester. Several autonomous options are available to compress/decompress repair information, program/read fuses, and manipulate BISR chains. These options are explained in the AutonomousOption wrapper description.
fuse_box_access	Specifies that the BISR controller enables access to the fuse box using the TDI and TDO top-level 1149.1 TAP ports. The two valid operations in this mode are <i>program</i> and <i>read</i> . These are explained in the FuseBoxAccessOptions wrapper description.

Examples

Examples for the various BISR run modes can be found at the end of the [BisrChainAccessOptions](#), [AutonomousOption](#), and [AutonomousOption](#) wrapper descriptions.

Controller

Specifies the inclusion of a memory BISR controller into the test patterns.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBISR {
                Controller(icl_instance_name) {
                    power_domain_group_labels      : pdg_label, ... ;
                    test_time_multiplier          : real ; // default 1
                    fuse_box_write_duration       : time ; // default 7ms
                    fuse_box_read_duration        : time ; // default 200ns
                    fuse_box_init_duration        : time // default 1us
                    inhibit_buffer_to_fuse_tranfer : on | off ;
                    AutonomousOptions {
                    }
                    BisrChainAccessOptions {
                    }
                    FuseBoxAccessOptions {
                    }
                }
            }
        }
    }
}
```

Description

The Controller wrapper specifies the inclusion of a memory BISR controller into the test patterns and allows specifying properties that are applicable to all run modes selectable in the [MemoryBist](#) wrapper such as the selection of power domain groups. Options specific to the selected run mode are part of the corresponding wrapper.

Arguments

- *icl_instance_name*

Specifies the instance name of a module found in an ICL model. The ICL instance name corresponds to the instance of the module in the design at the time the BISR logic was inserted. The ICL instance name provides a fixed reference to this logic even if the circuit hierarchy might change during the layout process. At the block/sub-block level when the chain are not yet connected to a controller, the *icl_instance_name* is specified as “.”. and identifies the block/sub-block itself. At the chip top level, it identifies the BISR controller module. The list of available memory BIST controller instances can be found using this command:

```
get_icl_instances -filter
{tessent_instrument_type==mentor::memory_bisr &&
tessent_instrument_subtype==controller}
```

The [create_patterns_specification](#) command automates the creation of the Controller wrapper for each instance in the design.

- `power_domain_group_labels : pdg_label, ... ;`

A property that specifies the power domain group(s) on which to perform the BISR operations. The default value is an asterisk (*) which is equivalent to enumerating all power domain group labels, including the default group assigned to all memories that were not explicitly assigned a label. This special group is designated with a hyphen (-).

Note the following usage conditions:

If these settings apply:	Then the power_domain_group_labels setting must be:
<code>MemoryBisr/run_mode : bisr_chain_access</code>	An asterisk (*) or One or more power domain group label(s)
<code>MemoryBisr/run_mode: autonomous</code> <code>MemoryBisr/Controller/AutonomousOptions/operation : verify_fuse_box</code> or <code>MemoryBisr/Controller/AutonomousOptions/operation : power_up_emulation</code> or <code>MemoryBisr/Controller/AutonomousOptions/operation : clear_bisr_chain</code>	An asterisk (*) or One or more power domain group label(s)
<code>MemoryBisr/run_mode: autonomous</code> <code>MemoryBisr/Controller/AutonomousOptions/operation : self_fuse_box_program</code>	An asterisk (*) or all power domain group labels

- `test_time_multiplier : real ;`

A property that specifies a multiplier to extend or shorten the memory BISR run time. The final run time is equal to the original test time (as calculated by Tesson Shell) multiplied by the `test_time_multiplier` factor. The default is 1.

This property is typically used when the test clock runs slower than expected by BISR. As a result, the memory BISR controller may not have completed its test within the allocated time when polled by the ATE. The `test_time_multiplier` property thus extends the computed test time; this is usually as a temporary measure (until the clock periods described in the [Patterns/ClockPeriods](#) wrapper is used).

- `fuse_box_write_duration : time ;`

A property that controls the duration of the pause cycle when the BISR controller is writing into the fuse box. This property is rarely used and overrides the value already specified by

the [AdvancedOptions/write_duration](#) property of the fuse box interface. The fuse box data sheet should provide the fuse write duration delay. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box write duration default is 7ms.

- `fuse_box_read_duration : time ;`

A property that controls the duration of a read access from the fuse box. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box read duration default is 200ns.

- `fuse_box_init_duration : time ;`

Specifies the initial delay required prior to accessing the fuse box. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box init duration default is 1 us.

- `inhibit_buffer_to_fuse_transfer : on | off ;`

A property that prevents the automatic programming of the fuse values that are stored in a buffer located in the fuse box interface at the end of a `SelfFuseBoxProgram` or `FuseBoxAccess` programming test step. If the `inhibit_buffer_to_fuse_transfer` property is set to “on” inside a `self_fuse_box_program` or `fuse_box_access` programming `TestStep` wrapper, the contents of the fuse box interface buffer is not transferred to the fuse box (specifically, fuse programming is not executed). Setting this property to on enables additional fuse values to be written to the buffer in subsequent test steps before final programming is executed.

Examples

Examples for the various BISR options can be found at the end of the [BisrChainAccessOptions](#), [AutonomousOption](#), and [AutonomousOption](#) wrapper descriptions.

AutonomousOption

Sets options that control the autonomous mode of the BISR controller.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBisr {
                Controller(instance) {
                    AutonomousOptions {
                        operation : mode ;
                        enable_bira_capture : on | off ;
                        select_read_buffer : on | off | auto ;
                        max_repair_count : integer | all ;
                    }
                }
            }
        }
    }
}
```

Description

The AutonomousOptions wrapper sets options that control the autonomous mode of the BISR controller. In Autonomous mode, the controller operation does not require any data patterns from the tester and the state of the GO and DONE outputs is updated at the end of the operation.

Several autonomous options are available to compress and decompress repair information, program and read fuses, and manipulate BISR chains. The content of the AutonomousOptions wrapper is used when you specify the MemoryBisr/run_mode property as “autonomous”.

Arguments

- **operation : mode ;**

A property that specifies one of the following operation modes:

calculate_bisr_chain_length	In this mode, the BISR controller measures the length of the selected BISR chains by shifting a single “1” through the chains. The fuse box controller counts the number of shifts needed for the “1” to propagate through the chains and initializes the BISR chain length registers with the calculated value. These registers are also initialized using the same method when running the power_up_emulation operation. The fuse box is not read from for this operation. The initialization of the BISR chain length registers must be executed before running the following autonomous mode operations: rotate_bisr_chain, self_fuse_box_program, and verify_fuse_box.
-----------------------------	---

clear_bisr_chain	Clears all flip-flops in the BISR chain without calculating the BISR chain length. This enables quick initialization of the memory repair inputs when performing memory BIST verification.
power_up_emulation	Simulates a power-up event. During this mode, the BISR controller first calculates the BISR chain length using the method outlined for calculate_bisr_chain_length and then reads and decompresses the repair information from the fuse box into the selected BISR chain(s). All inputs to the BISR controller are provided by IJTAG in this mode. The functional inputs of the BISR controller (clock, repair enable and PDGenable signals) are not used in that mode and can be in any state.
rotate_bisr_chain	Performs a BISR chain rotation autonomously. This operation is used to shift the contents of the BISR chain inside memories with internal repair registers. When the enable_bira_capture property is set to on, a BIRA-to-BISR transfer is performed prior to the BISR chain rotation.
self_fuse_box_program	When the BISR chain contains the repair information for all memories, the BISR controller rotates the BISR chain, compresses its content, and writes the compressed values to the fuse box. If the GO and DONE signals of the BISR controller are high when the test step completes, the initial BISR chain content is preserved. When multiple power domain groups are present, their BISR chains are processed in a single controller run.
verify_fuse_box	Verifies the decompressed content of the fuse box against the content of the BISR chain, one bit at a time. The BISR chain is shifted as the decompressed content of the fuse box is shifted in and compared with the BISR chain output. The BISR controller GO signal goes high at the end of the test if the decompressed data from the fuse box exactly matches the content of the BISR chain.

- enable_bira_capture : on | off ;

A property that enables BISR registers to capture the repair information contained in the corresponding BIRA registers in the rotate_bisr_chain mode. When set to on, the BIRA value is loaded into the BISR register on the first BISR clock cycle before shifting the BISR chain. When set to off, loading of the BIRA values is disabled.

- select_read_buffer : on | off | auto ;

This property is only used if the [MemoryBisr/memory_repair_loading_method](#) parameter is set to “from_read_buffer”. The select_read_buffer property is used to select between serial or parallel loading of the BISR chains during the power-up emulation autonomous mode of the fuse box controller. When selecting the read buffer output, the BISR chain initialization

is much faster than the serial method since the repair values are driven to the memory repair ports using parallel connections. If enable_bira_capture is enabled the parallel repair values are also captured into the BISR registers for validation purposes. When select_read_buffer is set to “off”, the repair values from the fuse box are shifted serially in the BISR registers.

The default “auto” value resolves to “on” if the fuse box controller does support fast BISR loading and the run_mode for the current test step is “power_up_emulation”. Otherwise, the “auto” value resolves to “off”.

- max_repair_count : *integer* | all ;

A property that specifies the maximum number of repairs your test bench will perform. This property is used only in simulation mode with fault injection. Because you know how many faults you injected and how many repairs this represents, you can specify max_repair_count to reduce the simulation time. Valid integer values are 1 or greater. A setting of “all” means that the runtime is calculated as the time needed to write 50% of the fuse box locations, assuming the other 50% will be zeros. A *test_time_multiplier* : *real* ; property can be specified with a value varying between 1 and 2 to adjust the test time to adjust the maximum percentage of programmed fuses, although this should rarely be necessary.

Examples

The following example will cause the fuse box controller to find the repair information for the specified power domain in the fuse box, read it, decompress it and load the result in the BISR chain associated to power domain named “PDG_A”. The BISR chain length is also calculated.

```
TestStep(power_up_emulation_step) {
    MemoryBisr {
        run_mode : autonomous ;
        Controller(FB_INST1) {
            power_domain_group_labels : PDG_A ;
            AutonomousOptions {
                operation : power_up_emulation ;
            }
        }
    }
}
```

BisrChainAccessOptions

Specifies the options that control BISR chain operations.

Usage

```
PatternsSpecification(design_name,design_id, pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBisr {
                Controller(icl_instance_name) {
                    BisrChainAccessOptions {
                        enable_rotation : on | off ;
                        enable_bira_capture : on | off ;
                        select_read_buffer : on | off ;
                        select_bisr_registers : internal | external ;
                        default_read_value : pattern ;
                        default_write_value : pattern ;
                        BisrRegisterAccessOptions(icl_instance_name) {
                            read_value(range) : pattern ;
                            write_value(range) : pattern ;
                        }
                    }
                }
            }
        }
    }
}
```

Description

The BisrChainAccessOptions wrapper specifies the options that control BISR chain operations.

This wrapper can be used at all design levels (specifically, top, block or sub-block level). To use the BisrChainAccessOptions wrapper, you must specify [MemoryBisr/run_mode](#) : `bisr_chain_access`.

You specify access to individual BISR registers using the BisrRegisterAccessOptions. Any `read_value` or `write_value` specified in this wrapper overrides the `default_read_value` and `default_write_value` specified in the BisrChainAccessOptions wrapper.

Arguments

- `enable_rotation : on | off` ;

A property that enables the rotation of the BISR selected chain(s). This property is typically used in combination with `enable_bira_capture : on` and `select_bisr_registers : external` to transfer repair information from BIRA registers to external and internal BISR registers. This property can only be specified when the chains are not yet connected to a controller. When you are at a level where the chains are connected to a controller, rotation of the BISR chains must be performed using the `rotate_bisr_chain` autonomous mode described in the [AutonomousOption](#) wrapper.

- `enable_bira_capture : on | off ;`

A property that enables BISR registers to capture the repair information contained in the corresponding BIRA registers. When set to on, the BIRA values are loaded into the BISR registers on the first BISR clock cycle before shifting the BISR chain(s). When set to off, loading of the BIRA values is disabled. When set to on, `select_bisr_registers` should be set to external. The `enable_rotation` property can be set to on to enable the transfer of BIRA registers to internal BISR registers, or it can be set to off if the objective is to simply inspect the values contained in the BIRA registers.

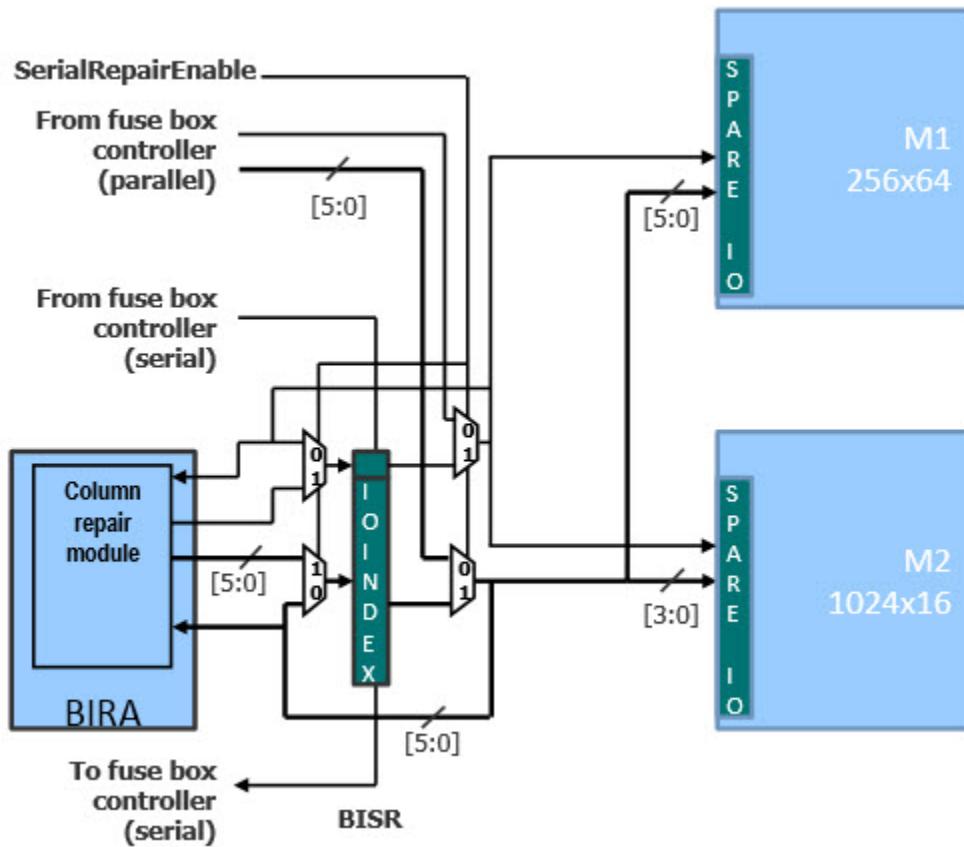
- `select_read_buffer : on | off ;`

This property is only used if the [MemoryBisr/memory_repair_loading_method](#) parameter is set to “from_read_buffer”.

When `select_read_buffer` is set to on, the `SerialRepairEnable` signal driving the multiplexers shown in [Figure 10-59](#) below is set to “0”. This will allow the parallel inputs driven by the fuse box read buffer output to reach the memory repair ports. Additionally, if the `enable_bira_capture` property is enabled, the parallel repair inputs will also be captured inside the BISR registers at the beginning of the shift cycle and scanned out.

When `select_read_buffer` is set to off, the `SerialRepairEnable` signal driving the multiplexers is set to “1”. This will allow the repair values from the BISR register outputs to drive the memory repair ports. When `enable_bira_capture` is enabled in this case, the values from the BIRA engine will be captured inside the BISR registers at the beginning of the shift cycle.

Figure 10-59. Fast BISR Muxing



When select_read_buffer is enabled at a block level, the primary inputs of the block are captured. For block level, an iProc is needed that provides iForcePort statements on the primary inputs of the block to emulate read buffer values. This prevents unknown values from being captured into the BISR registers. The iProc shown below provides an example implementation:

```
iTopProc ReadBuffer{} {
    iForcePort ReadBufferParallelPortName 0
    iApply
}
```

- `select_bisr_registers : internal | external ;`

A property that specifies whether the internal BISR register of memories with a serial BISR interface, or part of a shared bus, are scanned out during the BISR scan chain access. When set to internal, the property can be used to confirm the values actually loaded in internal BISR registers in a preceding test step.

- `default_read_value: pattern ;`

A property that sets default compare values for all BISR registers of the selected BISR chain(s). The BisrRegisterAccessOptions/read_value property can be used to override the compare pattern for specific BISR registers. By default, the all_x pattern is selected

indicating that all bits of all BISR registers will be ignored until specified otherwise in the patterns specification. Similarly, the all_one and all_zero patterns indicate that all bits will be compared to 1 and 0 respectively. A binary bit pattern can also be specified for comparison. The leading_one pattern indicates that the bit of each BISR register closest to its serial output will be compared to 1 and all others to 0. This pattern is useful for verifying the correct operation of BISR chains. The valid values are:

<i>binary</i>	During an iRead operation, the BISR chain range is compared against the specified bit pattern.
<u>all_x</u>	During an iRead operation, the BISR chain values are not compared.
all_one	Compares all BISR register bits against all 1s during an iRead operation.
all_zero	Compares all BISR register bits against all 0s during an iRead operation.
leading_one	The leading one symbol corresponds to a value of 1 followed by 0s (100...00).

- default_write_value : *pattern*;

A property that sets default values to be scanned in all BISR registers of the selected BISR chain(s). The BisrRegisterAccessOptions/write_value property can be used to override the write value for specific bits. By default, the all_x pattern is selected indicating that none of the bits need to be set to a particular value. In that case, the last value scanned in during a BISR chain access will be used to set each bit. The all_one and all_zero patterns indicate that all bits will be set to 1 and 0 respectively. The bits can also be set to a specified binary bit pattern. The all_x pattern is used as the default because BISR chains are usually reset before being exercised for the first time. It is therefore not necessary to explicitly write a value in order to initialize all bits of the BISR chain to 0. The leading_one pattern indicates that the bit of each BISR register closest to its serial output will be set to 1 and all others to 0. This pattern is useful for verifying the correct operation of BISR chains.

<i>binary</i>	During an iWrite operation, the BISR chain range is written with the specified bit pattern.
<u>all_x</u>	During an iWrite operation, the test bench will scan in the same values of those that were previously shifted in this shift register.
all_one	Writes 1's in all BISR bits during an iWrite operation.
all_zero	Writes 0's in all BISR bits during an iWrite operation.
leading_one	The leading one symbol corresponds to a value of 1 followed by 0s (100..00).

- *icl_instance_name*

Specifies the instance name of one or several BISR registers found in an ICL model of the circuit. The selected BISR registers must be part of the selected BISR chain(s) identified by the power_domain_group_labels property. Multiple BISR registers can be specified using wildcards within the name. The ICL instance names correspond to the instance name of BISR registers in the design at the time the BISR logic was inserted. The ICL instance name provides a fixed reference to this logic even if the circuit hierarchy might change during the layout process. To report the available BISR registers, use the [report_bisr_repair_register_icl_instances](#) command.

- *read_value(range) : pattern;*

A property that sets an expected value on specific bits of the selected BISR registers. This property is used only for verification and serves no purpose in the manufacturing flow. The range entry represents a bit index or range of bit indices of BISR registers within the selected BISR chain(s). Bit 0 is closest to the serial output of the BISR register. Entries for range are:

range	Description
<i>integer</i>	Reads the specified bit index of the BISR chain.
<i>integer:integer</i>	Reads the range of bit indices specified by the two integers.
<i>all_bits</i>	Reads all bits of the BISR registers within the selected chain(s).
<i>max_index:integer</i>	<i>max_index</i> is replaced by the most significant bit index value of the BISR registers within the selected chain(s). Reads the range of bit indices between the most significant bit and the index specified by <i>integer</i> .
<i>max_index</i>	Reads the most significant bit of the selected BISR chain(s).

The binary value for pattern is a bit pattern of a determined length—for example 8'b10001101. The bit length defined must match the index length defined by range or an error is generated. The all_zero pattern indicates that all bits of the selected BISR registers will be compared to 0. Similarly, the all_one and all_x patterns indicate that selected BISR register bits will be compared to 1 and x respectively. The leading_one pattern indicates that the bit within a range of BISR registers closest to its serial output will be compared to 1 and all others to 0. Values for pattern are:

<i>binary</i>	During an iRead operation, the BISR chain range is compared against the specified bit pattern. The length of range and pattern must match or an error is generated.
---------------	---

all_x	During an iRead operation, the BISR chain values are not compared.
all_one	Compares the BISR register bits against all 1s during an iRead operation.
all_zero	Compares the BISR register bits against all 0s during an iRead operation.
leading_one	During an iRead operation, compares the BISR register bits against a value of 1 followed by 0s (100...00) starting at the bit within the range closest to the BISR chain serial output.

- `write_value(range) : binary;`

A property that sets values to be scanned into specific bits of the selected BISR registers. For memories with a serial BISR interface, both the internal and external BISR registers are set to the same value. This property is used only for verification and serves no purpose in the manufacturing flow. The range entry represents a bit index or range of bit indices of BISR registers within the selected BISR chain(s). Bit 0 is closest to the serial output of the BISR register. Entries for range are:

range	Description
<i>integer</i>	Writes the specified bit index of the BISR chain.
<i>integer:integer</i>	Writes the range of bit indices specified by the two integers.
all_bits	Writes all bits of the BISR registers within the selected chain(s).
<i>max_index:integer</i>	<i>max_index</i> is replaced by the most significant bit index value of the BISR registers within the selected chain(s). Writes the range of bit indices between the most significant bit and the index specified by <i>integer</i> .
<i>max_index</i>	Writes the most significant bit of the selected BISR chain(s).

The binary value for pattern is a bit pattern of a determined length—for example 8'b10001101. The bit length defined must match the length index defined by range or an error is generated. The all_zero and all_one patterns indicate that all bits of the selected BISR registers will be written with 0s and 1s respectively. The all_x patterns indicate that the selected BISR register bits will be written with the same values that were previously

shifted in. The `leading_one` pattern indicates that the bit within the range of BISR registers closest to its serial output will be compared to 1 and all others to 0. Values for patterns are:

<code>binary</code>	During an iWrite operation, the BISR chain range is written with the specified bit pattern. The length of range and pattern must match or an error is generated.
<code>all_x</code>	During an iWrite operation, the test bench will scan in the same values of that were previously shifted in this BISR chain range.
<code>all_one</code>	Writes 1s in the BISR chain range during an iWrite operation.
<code>all_zero</code>	Writes 0s in the BISR chain range during an iWrite operation.
<code>leading_one</code>	During an iWrite operation, writes a value of 1 followed by 0s (100...00) starting at the bit within the range closest to the BISR chain serial output.

To use the `write_value` property, the [BisrChainAccessOptions/enable_rotation](#) property must be set to off.

Examples

Example 1

The following example has two TestStep wrappers. Both test steps operate on the BISR chain identified with label PDG_A. The `leading_one` pattern is loaded in all BISR registers in the first step and read back in the second one.

```
TestStep(write_leading_one) {
    MemoryBisr {
        run_mode      : bisr_chain_access ;
        Controller(FB_INST1) {
            power_domain_group_labels : PDG_A ;
            BisrChainAccessOptions {
                default_write_value   : leading_one ;
            }
        }
    }
}
TestStep(read_leading_one) {
    MemoryBisr {
        run_mode      : bisr_chain_access ;
        Controller(FB_INST1) {
            power_domain_group_labels : PDG_A ;
            BisrChainAccessOptions {
                default_read_value    : leading_one ;
            }
        }
    }
}
```

Example 2

The following example has two TestStep wrappers. Both test steps operate on the BISR chain identified with label PDG_A. The first step loads a value of 0 in all bits of the BISR chain except for a portion of the specified BISR register. The second step confirms that all values have been loaded correctly and the BISR chain is functional. These steps could be followed by another one running memory BIST to verify that the repair solution loaded in the BISR register is valid.

```
TestStep(write_selected_BISR_reg) {
    MemoryBisr {
        run_mode : bisr_chain_access ;
        Controller(FB_INST1) {
            power_domain_group_labels : PDG_A ;
            BisrChainAccessOptions {
                default_write_value : all_zero ;
                BisrRegisterAccessOptions (BISR_reg_inst3) {
                    write_value(15:8): 8'b10001101 ;
                }
            }
        }
    }
}

TestStep(read_selected_BISR_reg) {
    MemoryBisr {
        run_mode : bisr_chain_access ;
        Controller(FB_INST1) {
            power_domain_group_labels : PDG_A ;
            BisrChainAccessOptions {
                default_read_value : all_zero ;
                BisrRegisterAccessOptions (BISR_reg_inst3) {
                    read_value(15:8): 8'b10001101 ;
                }
            }
        }
    }
}
```

FuseBoxAccessOptions

Specifies the options used to read and write (or program) fuses to the fuse box.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBISR {
                Controller(instance) {
                    FuseBoxAccessOptions {
                        operation : program | read ;
                        read_address(range) : 0 | 1 | X ; //repeatable
                        write_address : range ; //repeatable
                    }
                }
            }
        }
    }
}
```

Description

The FuseBoxAccessOptions wrapper specifies the options used to read and write (program) fuses in the fuse box. Fuse addresses are scanned in using the ScanIn and ScanOut ports of the IJTAG network (which is typically a TAP controller at the chip level). To use the FuseBoxAccessOptions wrapper, you must specify MemoryBISR/run_mode : fuse_box_access.

Arguments

- **operation : program | read ;**

A property that specifies the type of access method the BISR controller executes on the fuse box. When set to read, the BISR controller executes a read operation on the fuse box according to the read_address property setting. When set to program, the BISR controller writes a logic 1 value to the address specified with the write_address property.

- **read_address(range) : 0 | 1 | X ;**

A property that reads the content at the specified fuse box address and compares the content against logic 0 or 1. To disable the compare, set the property to X. The default is to compare all addresses to 0 that correspond to the value of an unprogrammed fuse. The range specifies the address within the fuse box. The range can be a single address, an address range, or a comma-separated list of the two formats. All addresses must be an integer varying from 0 to number_of_fuses as defined by the [AdvancedOptions/number_of_fuses_for_repair](#) property in the [DftSpecification](#). Its value is either explicitly defined or calculated by default to be (2address_bits - 1) where address_bits is the width of the address port specified in the FuseBoxInterface wrapper. The “max_address” and “max_repair_address” strings can be used to specify a range. The “max_address” string represents the last address of the fuse box whereas the “max_repair_address” string represents the last address dedicated to memory repair which is determined by the value of number_of_fuses_for_repair defined in the [AdvancedOptions/FuseBoxOptions](#) wrapper or calculated by default to be the same as

number_of_fuses. For example, the range “max_address: 0” selects all bits of the fuse box whereas “max_repair_address: 0” selects the lower portion of the fuse box dedicated to memory repair. The read_address property can be repeated several times within the FuseBoxAccessOptions wrapper. If an address is specified in more than one property, the last specification will prevail. To use this property, the operation property must be set to read.

- write_address : *range* ;

A property that writes logic 1 into the specified fuse box address which is an integer. The range can be a single address, an address range, or a comma-separated list of the two formats. All addresses must be an integer varying from 0 to number_of_fuses as defined by the [AdvancedOptions/number_of_fuses_for_repair](#) property in the [DftSpecification](#). Its value is either explicitly defined or calculated by default to be (2address_bits -1) where address_bits is the width of the address port specified in the FuseBoxInterface wrapper. The “max_address” and “max_repair_address” strings can be used to specify a range. The “max_address” string represents the last address of the fuse box whereas the “max_repair_address” string represents the last address dedicated to memory repair which is determined by the value of number_of_fuses_for_repair defined in the [AdvancedOptions/FuseBoxOptions](#) wrapper or calculated by default to be the same as number_of_fuses. For example, the range “max_address: 0” selects all bits of the fuse box whereas “max_repair_address: 0” selects the lower portion of the fuse box dedicated to memory repair. The write_address property can be repeated several times within the FuseBoxAccessOptions wrapper. To use this property, the operation property must be set to program.

Examples

The following example has three TestStep wrappers. The first one verifies that the fuse box has never been written before (that is, all fuses have a value of 0). The second one writes 1 in a few selected locations. The third TestStep reads back a portion of the fuse box and verifies that the locations written in the second TestStep contain the value 1.

```
TestStep(read_all0s) {
    MemoryBisr {
        run_mode : fuse_box_access ;
        Controller(FB_INST1) {
            FuseBoxAccessOptions {
                // Not necessary to specify read_address.
                // Default is to read all addresses and compare to 0.
                // Equivalent to specifying read_address(max_address:0):0 ;
                operation : read ;
            }
        }
    }
}
TestStep(write_selected_locations) {
    MemoryBisr {
        run_mode : fuse_box_access ;
        Controller(FB_INST1) {
            FuseBoxAccessOptions {
                operation : program ;
                // Write a 1 to 7 locations
                // 5:9 is equivalent to 5, 6, 7, 8, 9
                write_address : 2, 5:9, 12 ;
            }
        }
    }
}
TestStep(verify_first_16_locations) {
    MemoryBisr {
        run_mode : fuse_box_access ;
        Controller(FB_INST1) {
            FuseBoxAccessOptions {
                operation : read ;
                // Restrict the address range to the first 16 addresses
                // Set the expected value to 0
                read_address(15:0) : 0 ;
                // Override the expected value for the locations written in
                // previous step
                read_address(2, 5:9, 12) : 1 ;
            }
        }
    }
}
```

MemoryBist

Specifies the use of MemoryBist in the current TestStep.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                run_mode : mode ;
                parallel_retention_time : time | off ;
                reduced_address_count : on | off ;
                AdvancedOptions {
                    retention_test_phase : all_phases | start_to_pause |
                                            pause_to_pause | pause_to_end ;
                    memory_reset : on | off ;
                    preserve_bist_inputs : on | off ;
                }
                DiagnosisOptions {
                    extract_diagnosis_data : on | off | auto ;
                    preserve_fuse_register_values : on | off ;
                }
                Controller(instance) {
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the use of MemoryBist in the current TestStep.

This wrapper can be specified once for each TestStep wrapper. The MemoryBist specification consists of one AdvancedOptions wrapper, one DiagnosisOptions wrapper, and one or more Controller wrapper(s). These three wrappers can be specified in any order.

The repeatable Controller wrapper is used to define the inclusion of the given controller instance in the current TestStep, and allows specifying additional options that are specific to the instance. The properties specified outside the Controller wrapper apply to all Controllers.

The AdvancedOptions wrapper contains advanced options influencing how memory BIST is carried out. The DiagnosisOptions wrapper defines options related to memory BIST diagnosis.

Arguments

- `run_mode : mode ;`

A property that specifies the run mode to use with the controllers. The valid modes and their meaning are:

<u>hw_default</u>	Specifies that the memory BIST controllers use the default (hardcoded) settings. This run mode requires the least amount of configuration for the memory BIST controllers to launch tests.
<code>run_time_prog</code>	Specifies that the memory BIST controllers run the BIST test using the scanned-in settings prior to test execution. Individual memory BIST controllers are first accessed and configured before they all execute tests in parallel.
<code>increment_failure_limit</code>	Automatically advances the optional failure limit counter. This mode is useful for creating Stop-On-Error test patterns for bitmap applications.
<code>check_repair_needed</code>	Specifies that the repair analysis status registers need to be examined after BIST completes. The status registers report the pair status of the memories with built-in redundancy. This mode requires at least one tested memory to have repair analysis features defined in its Memory library description. This is indicated by the presence of a RedundancyAnalysis wrapper. See “ Tessent Core Description ” on page 3755.

- `parallel_retention_time : time | off ;`

A property that specifies the retention pause between algorithm sub-phases when performing parallel static retention testing. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. Specifying a time value for a controller testing only ROM will result in an error.

- `reduced_address_count : on | off ;`

A property that enables the memory BIST controller to be run on the four corners of the common memory address space. This property is useful for checking the proper functionality of the BIST controller without having to simulate the test of the entire memory space. Static timing analysis is required in the full chip context to verify that the timing between the controllers and the memories is valid. Note that the full address space should be simulated on the assembly module before it is merged into the chip.

This property supports the `hw_default` or `run_time_prog` run modes.

Patterns created for signoff usage with the [create_patterns_specification](#) command set the `reduced_address_count` property to on for memory BIST controllers with RAMs only. Memory BIST controllers with ROMs will ignore this property and execute a full address count.

When this property is enabled and the memory BIST controller contains a step with multiple memories, the controller runs all algorithm phases of the test on just the following four addresses:

- RowMin, ColMin
- RowMax, ColMin
- RowMin, ColMax
- RowMax, ColMax

If the memory has only row address bits, the reduced address range would be these two locations:

- RowMin
- RowMax
- AdvancedOptions/retention_test_phase : all_phases | start_to_pause | pause_to_pause | pause_to_end ;

A property that specifies whether the memory BIST controller applies the entire selected library algorithm or only a sub-phase of it. Specifying a phase value other than all_phases for a controller only testing ROM will result in an error.

This property is typically not used because specifying a time value with the parallel_retention_time property described above automates the sequential execution of the start_to_pause, pause_to_pause, and pause_to_end phases with the specified pause time inserted between them. This retention_test_phase enables you to define the step explicitly into three TestSteps where each TestStep runs a different phase. Valid values are as follows:

all_phases	Indicates that the controller applies the entire selected algorithm.
start_to_pause	Indicates that the controller applies only the portion of the selected algorithm that ends at the first static retention test pause. This value corresponds to the application of the checkerboard pattern.
pause_to_pause	Indicates that the controller applies only the portion of the selected algorithm that runs from the first static retention pause to the second static retention test pause. This value corresponds to the application of the inverse checkerboard pattern.
pause_to_end	Indicates that the controller applies only the portion of the selected algorithm that runs from the second static retention pause and onward.

- AdvancedOptions/memory_reset : on | off ;
- A property that instructs memory BIST controllers to only clear the memories and to skip the remaining phases of the library algorithm. The controllers clear memories by writing a value of 0 to all memory locations. You can only set this property to on when the algorithms used are limited to the following: SMarch, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, or SMarchCHKBvcd. Specifying an on value when other algorithms are used results in an error. Specifying an off value causes the full algorithm to be executed. The

memories are populated with zeros at the end of the algorithm since the last phase executed writes a value of 0 to all memory locations.

- AdvancedOptions/preserve_bist_inputs : on | off ;

A property that specifies that the memory interface multiplexers will continue to select the BIST inputs during and after the test has finished. Memories are under functional control except for the memories undergoing test by the test algorithm. When set to on, the memories will remain under BIST control for the duration of the TestStep until the property is set to off in a subsequent TestStep. One application of this property is to maintain the BIST inputs to the memory between pattern TestSteps. Another application of this property is to allow the memory to be refreshed by the controller upon completion of all algorithms in the test.

- DiagnosisOptions/extract_diagnosis_data : on | off | auto ;

A property that enables the sampling of internal registers of the memory BIST controller after the BIST run is completed. This property is useful when the Stop-On-Nth-Error diagnosis feature is enabled using failure_limit. The internal register values are only meaningful if the controller has stopped at the requested failure number and the DONE status signal is 0. Furthermore, the internal register values require post-processing to provide diagnostic information on the detected failures. The default value of auto resolves to on when failure_limit is not off, otherwise it resolves to off.

- DiagnosisOptions/preserve_fuse_register_values : on | off ;

A property that specifies if the repair analysis registers preserve their values from a previous test step. This inhibits the loading of the BISR register values into the repair analysis registers when a new test step begins. This property supports the hw_default or run_time_prog run modes.

Examples

The following example runs a memory BIST in reduced address space.

```
PatternsSpecification(top,rtl,signoff) {
    Patterns(MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep(mem_test) {
            MemoryBist {
                run_mode : run_time_prog ;
                reduced_address_count : on ;
                Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
                    ...
                }
            }
        }
    }
}
```

Controller

Specifies the inclusion of a MemoryBIST controller instance in the TestStep.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    parallel_retention_group : integer ;
                    AlgorithmSetupOverrides {
                    }
                    AdvancedOptions {
                    }
                    RepairOptions {
                    }
                    DiagnosisOptions {
                    }
                    MemoryInterface(memory_id) {
                    }
                    DramOptions {
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the inclusion of a MemoryBIST controller instance in the TestStep.

This wrapper is repeated for each controller instance that is to be run in this TestStep. All MemoryBIST controller instances are set up serially but run in parallel.

Arguments

- *instance*

A string that specifies the ICL instance name of the MemoryBIST controller. The list of available MemoryBIST controller instances can be found using this command:

```
get_icl_instances -filter
{tessent_instrument_type==mentor::memory_bist &&
tessent_instrument_subtype==controller}
```

The [create_patterns_specification](#) command automates the creation of the Controller wrapper for each instance in the design.

- *parallel_retention_group : integer ;*

A property that specifies the group number associated with the controller. The group number is a numeric label for the controller group. Each controller group consists of all controllers with the same parallel_retention_group property value. When performing a parallel static retention test on controller groups, the controller groups are run sequentially

so as to minimize the peak power requirement while maintaining the ability to bring all controllers from all groups to the pause state and perform a single retention pause and, optionally, a single voltage bump for all memories in the chip.

The controller group execution order is determined from the order of appearance of the group numbers. The following usage conditions apply:

- This property is meaningful only if the parallel_retention_time property in the MemoryBist wrapper specifies a time greater than zero.
- When performing a parallel static retention test on controller groups, all three algorithm sub-phases (start_to_pause, pause_to_pause, and pause_to_end) are applied. In this case, the AdvancedOptions/retention_test_phase property in the MemoryBist wrapper is ignored.

Examples

The following example assigns the first MemoryBIST controller to a group number 0. It will thus be enabled first for a retention test of 1 second:

```
TestStep(mem_test) {
    MemoryBist {
        run_mode : hw_default ;
        parallel_retention_time : 1s ;
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
            parallel_retention_group : 0 ;
            (...)
        }
    }
}
```

AlgorithmSetupOverrides

The AlgorithmSetupOverrides wrapper groups the DataGenerator and AddressGenerator wrappers, which define the initial values of the address and data pattern prior to the execution of the test algorithm.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    AlgorithmSetupOverrides {
                        load_counter_a_start_count : int ; //defalut: 0
                        AddressGenerator {
                            AddressRegisterA | AddressRegisterB {
                                load_bank_address : binary_address | auto ;
                                load_column_address : binary_address | auto ;
                                load_row_address : binary_address | auto ;
                            }
                        }
                        DataGenerator {
                            load_write_data : binary_data | auto ;
                            load_expect_data : binary_data | auto ;
                            invert_data_with_row_bit : r[x] | none ;
                            invert_data_with_column_bit : c[x] | none ;
                        }
                    }
                }
            }
        }
    }
}
```

Description

The AlgorithmSetupOverrides wrapper groups the DataGenerator and AddressGenerator wrappers, which require initialization prior to execution of the algorithm microprogram. The specified values override the values defined in the selected algorithm.

Using the AlgorithmSetupOverrides wrapper requires selecting an algorithm using the `apply_algorithm` property inside the `MemoryBist/Controller/AdvancedOptions` wrapper of the `PatternsSpecification`. Additional usage conditions apply to the `AddressGenerator` wrapper. Changing the initial value of any address register or of `CounterA` require setting the `AdvancedOptions/functional_debug_mode` property to on.

For more information, see the “[Functional Debug Memory Access](#)” appendix in *Tessent MemoryBIST User’s Manual for Use with Tessent Shell*.

Arguments

- `load_counter_a_start_count : int ;`
An optional property that specifies the startcount or starting count value for the CounterA general purpose module. The default value is “0”. The integer specified for `load_counter_a_start_count` must be in the range [0:2ⁿ-1], where *n* is the number of bits in CounterA, specified by the `counter_a_bits` property in the DftSpecification/MemoryBist/Controller/[AlgorithmResourceOptions](#) wrapper.
- `AddressGenerator/AddressRegisterA | B/load_bank_address : binary_address | auto ;`
An optional property that specifies a binary value to be loaded into the *BankAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *BankAddress* is the initial value of the bank address prior to execution of the microprogram. The value set with this property overrides the `load_bank_address` value defined in the memory library [Algorithm](#) wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of bank address bits specified in the Core/Memory/[AddressCounter](#) wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting bank address of all memories tested in the controller step.
- `AddressGenerator/AddressRegisterA | B/load_column_address : binary_address | auto ;`
An optional property that specifies a binary value to be loaded into the *ColumnAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *ColumnAddress* is the initial value of the column address prior to execution of the microprogram. The value set with this property overrides the `load_column_address` value defined in the memory library [Algorithm](#) wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of column address bits specified in the Core/Memory/[AddressCounter](#) wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting column address of all memories tested in the controller step.
- `AddressGenerator/AddressRegisterA | B/load_row_address : binary_address | auto ;`
An optional property that specifies a binary value to be loaded into the *RowAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *RowAddress* is the initial value of the row address prior to execution of the microprogram. The value set with this property overrides the `load_row_address` value defined in the memory library [Algorithm](#) wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of row address bits specified in the Core/Memory/[AddressCounter](#) wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting row address of all memories tested in the controller step.
- `DataGenerator/load_write_data : binary_data | auto ;`
An optional property that specifies a binary value to load into the *WriteData* register. The value loaded into the *WriteData* register is the initial value of the write data prior to

execution of the microprogram. The value loaded with this property overrides the `load_write_data` value defined in the memory library [DataGenerator](#) wrapper.

- `DataGenerator/load_expect_data : binary_data | auto ;`

An optional property that specifies a binary value to load into the *ExpectData* register. The value loaded into the *ExpectData* register is the initial value of the expect data prior to execution of the microprogram. The value loaded with this property overrides the `load_expect_data` value defined in the memory library [DataGenerator](#) wrapper.

- `DataGenerator/invert_data_with_row_bit : r[x] | none ;`

An optional property that enables you to specify a row address bit that will invert the applied write and expect data registers. The applied write and expect data registers are inverted when the specified row address bit is a “1”, and not inverted when the row address bit is a “0”. The specified value overrides the value defined in the selected algorithm. The default value of “none” specifies that no row bit is selected to invert the write and expect data registers. The specified `r[x]` index must select a valid row address bit where `x` is in the range of “0” to the number of row address bits, minus one. The number of row address bits is specified in the Core/Memory/[AddressCounter](#) wrapper of the memory library file.

- `DataGenerator/invert_data_with_column_bit : c[x] | none ;`

An optional property that enables you to specify a column address bit that will invert the applied write and expect data registers. The applied write and expect data registers are inverted when the specified column address bit is a “1”, and not inverted when the column address bit is a “0”. The specified value overrides the value defined in the selected algorithm. The default value of “none” specifies that no column bit is selected to invert the write and expect data registers. The specified `c[x]` index must select a valid column address bit where `x` is in the range of “0” to the number of column address bits, minus one. The number of column address bits is specified in the Core/Memory/[AddressCounter](#) wrapper of the memory library file.

Examples

For examples and further information, see the “[Functional Debug Memory Access](#)” appendix in *Tessent MemoryBIST User’s Manual for Use with Tessent Shell*.

AdvancedOptions

Specifies advanced test options for the selected memory BIST controller instance.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    AdvancedOptions {
                        apply_algorithm : algo_name ;
                        apply_operation_set : opset_name ;
                        enable_memory_list : memory_id, ... ;
                        freeze_step : off | integer ;
                        freeze_test_port : off | integer ;
                        functional_debug_mode : on | off ;
                        test_time_multiplier : real ;
                        test_execution_cycles : integer | auto ;
                        MemoryClusterOptions {
                            configuration_data(interface_id) : binary | auto;
                        }
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that specifies advanced test options for the selected memory BIST controller instance.

The AdvancedOptions wrapper includes the MemoryClusterOptions wrapper which provides additional options specific to Shared Bus memory clusters.

Arguments

- `apply_algorithm : algo_name ;`

A property that selects a test algorithm to apply to one or more memory BIST controller steps. The specified algorithm overrides the default algorithm associated with the controller step.

- The algorithm name must match either a Mentor Graphics library built-in algorithm name or a user-defined algorithm name that was built into the memory BIST controller in the DFT insertion phase. For the soft-programmable controller, the algorithm name may also match a user-defined algorithm specified in the [MemoryOperationsSpecification](#) that was loaded prior to pattern generation.
- This property requires setting the MemoryBist/run_mode property to run_time_prog.

- `apply_operation_set : opset_name ;`

A property that selects a hard-coded operation set to apply to one or more memory BIST controller steps. The specified operation set overrides the default operation set associated with the controller step.

- The operation set name must match either a Mentor Graphics library built-in operation set name or a user-defined operation set name that was built into the memory BIST controller in the DFT insertion phase.
- This property requires setting the MemoryBist/run_mode property to run_time_prog.

- `enable_memory_list : memory_id, ... ;`

An optional property that specifies one or more memories to enable during parallel testing within a controller step. Selection of more than one memory must be specified as a comma separated list of memory ID's. If the property is not specified, all memories are enabled. A warning will be issued if all memories are not enabled for testing in the current pattern. The memory_id is a valid memory ID specified by the Step/MemoryInterface wrapper within the MemoryBist wrapper in the [DftSpecification](#) and must belong to the BIST controller step specified by the freeze_step property. The MemoryBist/run_mode property must be set to run_time_prog.

- `freeze_step : off | integer ;`

A property that specifies for the memory BIST controller to run only a single [Step](#) defined in the [DftSpecification](#). The freeze_step property enables you to diagnose specific failures. Controller steps are numbered sequentially starting with 0. To specify a single controller step, you must set the MemoryBist/run_mode property to run_time_prog. When the freeze_step property is off, the memory BIST controller tests across all steps implemented in the memory BIST controller, and the results are accumulated across the entire BIST run.

- `freeze_test_port : off | integer ;`

A property that specifies a test port on which to freeze the memory BIST controller so that you can diagnose specific failures. At least one memory in the [TestStep](#) wrapper much have multiple test ports. To use this property, you must set the MemoryBist/run_mode property to run_time_prog. When the freeze_test_port property is off, the memory BIST controller executes the steps for all test port combinations implemented in the [MemoryBist](#) controller in the [DftSpecification](#), and the results are accumulated across the entire BIST run.

- `functional_debug_mode : on | off ;`

An optional property that enables a mode on the Tessent MemoryBIST controller to allow functional system debug. The functional_debug_mode property in the DftSpecification/ MemoryBist/Controller/[AdvancedOptions](#) wrapper must have been set to “on” to create necessary hardware for this property. For more information, see the “[Functional Debug Memory Access](#)” appendix in *Tessent MemoryBIST User’s Manual for Use with Tessent Shell*.

- `test_time_multiplier : real ;`

A property that specifies a multiplier to extend or shorten the memory BIST run time. The final run time is equal to the original test time (as calculated by Tesson Shell) multiplied by the `test_time_multiplier` factor. The default is 1.

This property is typically used when the test clock runs slower than expected by BIST. As a result, the memory BIST controller may not have completed its test within the allocated time when polled by the ATE. Specifying a value greater than 1 thus extends the computed test time; this is usually as a temporary measure (until the clock periods described in the [Patterns/ClockPeriods](#) wrapper is used).

- `test_execution_cycles : integer | auto ;`

A property that specifies the number of clock cycles required to execute the algorithm. A setting of `auto` means that the pattern generation tool automatically computes the number of test execution cycles. In most cases this is sufficient; however, this property gives you the option to specify an absolute number of clock cycles. This accommodates extreme cases where the automatically-generated number is not accurate. Note that, in general, correcting the calculated test time with the `test_time_multiplier` property is preferable because it is less error-prone.

- `MemoryClusterOptions/configuration_data(interface_id) : binary | auto ;`

A repeatable wrapper that specifies options related to Shared Bus memory clusters.

The `MemoryClusterOptions` wrapper is used in the `MemoryBist/Controller` wrapper.

A repeatable property that specifies a binary value to be applied to the interface port identified by `interface_id`. The binary value follows Verilog syntax (for example, `3'b101`). If specified, it overrides the default ConfigurationData value from the memory cluster library file (which corresponds to the default “`auto`” value).

The value specified for this property is used for all controller steps executed in the `TestStep`. If the value is not compatible with all steps, the `AdvancedOptions/freeze_step` must also be specified.

Examples

Example 1

The following example illustrates the use of the `apply_operation_set` property to override the operation set to syncWR. This new operation set is applied to all [Steps](#). The `apply_algorithm` property is not used which means the default algorithm to each Controller/Step is used.

```
PatternsSpecification(top,rtl,signoff) {
    Patterns(MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep(mem_test) {
            MemoryBist {
                run_mode : run_time_prog ;
                Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
                    AdvancedOptions {
                        apply_operation_set : SyncWR ;
                    }
                }
            }
        }
    }
}
```

Example 2

The following example illustrates how to specify multiple specific memories for parallel testing within a controller through the enable_memory_list property.

Two memories, M1 and M3, are specified as a Tcl list for set_config_value on the enable_memory_list property:

```
SETUP> set_config_value PatternsSpecification(top,rtl,signoff) /
    Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions/
    enable_memory_list [list M1 M3]
```

The specification can be examined with report_config_data on the AdvancedOptions wrapper:

```
SETUP> report_config_data PatternsSpecification(dashboard,rtl,signoff) /
    Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions

    AdvancedOptions {
        enable_memory_list : M1, M3;
    }
SETUP>
```

The specification properties for enable_memory_list can be retrieved as a Tcl list with the get_config_data command:

```
SETUP> get_config_value PatternsSpecification(dashboard,rtl,signoff) /
    Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions/
    enable_memory_list
M1 M3
SETUP>
```

DiagnosisOptions

Provides diagnosis-related options and properties.

Usage

```
PatternsSpecification(design_name,design_id, pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    DiagnosisOptions {
                        compare_go : on | off ;
                        compare_go_id : on | off ;
                        compare_memory_go : on | off ;
                        compare_misr : on | off ;
                        comparator_id_select : integer | all | none ;
                        StopOnErrorOptions {
                            failure_limit : integer | auto_increment | off ;
                            data_compare_time_slots : all | even | odd ;
                        }
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that provides diagnosis-related options and properties.

Arguments

- compare_go : on | off ;

A property that specifies that the pass/fail (GO) controller port is checked at the end of the test. This operation provides the composite pass/fail results from all comparators in the controller. You need to disable the compare on the global pass/fail (GO) signal for some of the patterns needed in the memory self repair flow as documented in the [Tessent MemoryBIST User's Manual](#). You also can use this for legacy controllers containing incorrect ROM signatures in the hardware. When you set this property to on, you typically set the compare_go_id property and the compare_misr property to on.

- compare_go_id : on | off ;

A property that specifies that the compare status registers (GO_ID bits) are scanned and examined at the end of the test. This operation identifies which comparator failed but not necessarily the step in which it failed. This property is meaningful when at least one RAM is tested by the memory BIST controller. It is ignored on controllers testing only memories of type ROM.

Note

 If you are not using the automation provided by Tessent SiliconInsight, you can find out which memory instance failed when the [Step/comparator_location](#) property was set to `shared_in_controller` in the [DftSpecification](#) by running the test with a `freeze_step` property set for each of the steps in the memory BIST configuration file.

When the [RepairOptions/check_repair_status](#) property is set to `non_repairable`, the behavior of the `compare_go_id` property is disabled for repairable memories with local comparators. The GO_ID registers for non-repairable memories or repairable memories with shared comparators are always sampled when `compare_go_id` is set to `on`.

- `compare_memory_go : on | off ;`

A property that enables examining the pass/fail status register of the individual memories at the end of the pattern execution. The per-memory status register must be present in the memory BIST controller hardware in order to use this property. Refer to the [DftSpecification DiagnosisOptions/go_status](#) property description for information on incorporating the per-memory status register.

- `compare_misr : on | off ;`

A property that specifies the ROM MISRs are to be examined at the end of the test. This property is ignored when the controller is testing only memories of type SRAM or DRAM. By default, the MISR signature stored in the TSDB will be used to compare the MISR register content. You may specify a different value using the [MemoryInterface/expected_rom_signature](#) property.

- `comparator_id_select : all | none | integer ;`

A property that specifies the individual status of the comparators in the controller to be routed to the compare status controller port. Valid values are as follows:

<code>all</code>	Selects the global controller comparator status signal.
<code>none</code>	Selects a constant logical 1, which implies the comparison result will be forced to consistently pass.
<code>integer</code>	A value between 1 and the number of shared comparators implemented in the controller. The number of shared comparators equals the maximum memory word size of all memories with the <code>Step/comparator_location</code> property set to <code>shared_in_controller</code> in the DftSpecification . The value of 1 corresponds to bit 0 of the memory data word.

These usage conditions apply:

- This property is only meaningful when the controller was generated with the [DiagnosisOptions/comparator_selection_mux](#) property set to `on` in the [DftSpecification](#).

- StopOnErrorOptions/failure_limit : *integer* | auto_increment | off ;

A property that enables Stop-On-Error (SOE) and specifies the number of failures that the memory BIST controller detects before stopping and capturing all relevant information.

This property has two usage scenarios. To learn more about how to use this property, refer to the “Stop-On-Nth-Error Approach” section of the “Tessent MemoryBIST Diagnosis” chapter in the [Tessent MemoryBIST User’s Manual](#). Valid values are:

<i>integer</i>	Specifies the number of failures that the controller detects before stopping and capturing. The value is loaded in the SOE counter and, if present, the failure limit counter.
auto_increment	Specifies that the stop-on-error counter is loaded with the value of the failure limit counter as it was set in a previous step, where run_mode was set to increment_failure_limit.
off	

These usage conditions apply:

- The DftSpecification/MemoryBist property [DiagnosisOptions/StopOnErrorOptions/failure_limit](#) was not set to off.
- When you set the failure_limit property to a non-zero value, you should set [DiagnosisOptions/extract_diagnosis_data](#) to on within the MemoryBist wrapper.

Note

 When the failure limit is reached and the test stops, it is normal for the Done bit to fail because the test did not successfully complete (it aborted at some point).

To run diagnosis on a single memory:

- Set the MemoryBist/[AdvancedOptions/freeze_step](#) property to the step number that corresponds to the memory to diagnose.
- Use the MemoryBist/[AdvancedOptions/enable_memory_list](#) property to select a single memory. It requires that the [AdvancedOptions/selective_parallel_memory_test](#) property in the DftSpecification/MemoryBist wrapper was not set to off.
- StopOnErrorOptions/data_compare_time_slots : all | even | odd ;

A property that selects the clock cycles in which StrobeDataOut is enabled during a read operation. This property should be used when running Stop-on-Nth-Error diagnosis on memories that have StrobeDataOut enabled on consecutive clock cycles during a read operation. The Stop-On-Error diagnosis mode needs two clock cycles upon detecting an error before freezing the state of the memory BIST controller and memory interfaces. This property guarantees that the state of the controller is properly preserved when using

operation sets containing a read operation with consecutive StrobeDataOut signals. Valid values are:

<u>all</u>	Enables StrobeDataOut on all clock cycles during Stop-On-Error Diagnosis mode.
even	Enables StrobeDataOut only on the odd clock cycles during Stop-On-Error Diagnosis mode.
odd	Enables StrobeDataOut only on the even clock cycles during Stop-On-Error Diagnosis mode.

To use this option, the failure_limit property value must be a positive integer or set to auto_increment.

Examples

The following example extracts diagnosis data from a memory, m11, tested in the third controller step (specifically, step 2). Since the memory is normally tested concurrently, two other memories, m12 and m13, are disabled when diagnosing the memory of interest:

```
PatternsSpecification(top,rtl,signoff) {
    Patterns(MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep(mem_test) {
            MemoryBist {
                run_mode : run_time_prog;
                reduced_address_count : off ;
                DiagnosisOptions {
                    extract_diagnosis_data : on ;
                }
                Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
                    AdvancedOptions {
                        freeze_step : 2 ;
                        enable_memory_list : m11 ;
                    }
                    DiagnosisOptions {
                        StopOnErrorHandler {
                            failure_limit : auto_increment ;
                        }
                    }
                }
            }
        }
    }
}
```

RepairOptions

Provides repair-related options and properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    RepairOptions {
                        check_repair_status      : non_repairable | on | off ;
                        extract_repair_fuse_map : on | off ;
                        spare_element_priority : row | column ;
                    }
                }
            }
        }
    }
}
```

Description

The RepairOptions wrapper specifies what to shift out and monitor with repairable memories, as well as the spare allocation method to use. For Memory wrapper details, see “[Tessent Core Description](#)” on page 3755.

Arguments

- RepairOptions/check_repair_status : non_repairable | on | off ;

A property that specifies that the repair analysis status registers are examined after the BIST run is completed. The status registers identify whether the memories with built-in redundancy require no repair, is repairable or is unrepairable. Valid values are:

on	Specifies that the repair analysis status registers are sampled and compared against the zero (0) value.
<u>off</u>	Specifies that the repair analysis status registers are not sampled.
non_repairable	Specifies that only the non-repairable flag of the status register is sampled.

At least one memory tested by the controller must have repair analysis defined in its Memory library description. This feature is useful for separating the bad or unrepairable devices from the good or repairable devices during manufacturing test. For further details on repair status, refer to the “[BIRA Repair Status Bits Checking](#)” topic in the *Tessent MemoryBIST User’s Manual For Use with Tessent Shell* manual.

- RepairOptions/extract_repair_fuse_map : on | off ;

A property that specifies that the repair analysis fuse registers are examined after the BIST run is completed. The fuse registers contain the repair information for memories with

built-in redundancy. When set to on, the repair analysis fuse registers are sampled and compared against the zero (0) value. Note that at least one memory tested by the controller must have repair analysis defined in its Memory library description. When set to off, the repair analysis fuse registers are not sampled.

Note

 The content of the fuse registers are lost after performing this operation. This option should be set to “off” if you plan to perform a BIRA to BISR transfer.

- RepairOptions/spare_element_priority : column | row ;

A property that specifies the type of spare element to be allocated first upon failures. This is used only for memories with redundant row and column elements. To find a repair solution for a faulty memory initially declared as unrepairable, try changing the repair strategy by selecting a different spare_element_priority setting. Valid values are:

column	<p>Specifies that all single-bit errors are repaired by redundant column or IO elements; multi-bit errors are automatically repaired by spare row elements. If the number of faults exceeds the number of redundant columns or IO elements, the remaining redundant row elements are allocated for these faults. This is the most flexible solution because it automatically allocates redundant row elements for multi-bit failures while allocating redundant column or IO elements for single-bit failures.</p> <p>A redundant column or IO element cannot repair a multi-bit failure. If only a redundant column or IO element is available when a multi-bit error is detected, the repair status will be unrepairable.</p>
row	<p>Specifies that redundant rows are allocated for all errors (single-bit and multi-bit). The redundant column/IO elements are only allocated for the remaining faults after all redundant rows are allocated.</p>

Examples

The following example extracts the repair status and repair solution from all repairable memories.

```
PatternsSpecification(top,rtl,signoff) {
    Patterns(MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep(mem_test) {
            MemoryBist {
                run_mode : run_time_prog ;
                Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
                    RepairOptions {
                        check_repair_status : on ;
                        extract_repair_fuse_map : on ;
                    }
                }
            }
        }
    }
}
```

MemoryInterface

Specifies test options for the memory associated with the selected memory interface.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    MemoryInterface(memory_id) { //repeatable
                        expected_rom_signature : binary | auto;
                        rom_content_file       : file_name;
                        comparator_id_select   : all | none | integer;
                    }
                }
            }
        }
    }
}
```

Description

A wrapper that specifies memory-specific options.

Arguments

- *memory_id*

A string that identifies a given memory interface. The id is one of the ids defined in the [Step/MemoryInterface](#) and [Step/ReusedMemoryInterface](#) wrappers inside the [DftSpecification](#).

- *expected_rom_signature* : `binary` | `auto` ;

A property that specifies the ROM signature value to be compared to. This property is useful when the ROM content has changed and the ROM signatures were not updated into the [instruments](#) directory in the TSDB (see “[Tessent Shell Data Base \(TSDB\)](#)” on page 3763). A value of “auto” selects the signature calculated from the file referenced by *rom_content_file*. Refer to the *rom_content_file* argument description for more information on how the ROM MISR signature is specified.

- *rom_content_file* : *file_name* ;

A property that specifies a ROM content file from which the MISR signature is calculated. If *expected_rom_signature* is set to “auto”, and the specified file is readable, the [process_patterns_specification](#) command will calculate the MISR signature from the file, and apply the new MISR signature in the simulation test bench or manufacturing test pattern.

When generating manufacturing test patterns, a ROM signatures file, named *design_name_design_id.rom_signatures_manufacturing*, will be stored next to the patterns specification file in the TSDB. The ROM signatures file contains the MISR signatures calculated by the [process_patterns_specification](#) command for each combination of the

ROM instance and its ROM content file. Note that the ROM signatures file is not created when generating simulation test benches.

The ROM signature file may be referenced if the ROM content file is unavailable during subsequent runs to generate manufacturing patterns or to perform diagnosis using Tesson SiliconInsight. The tool can automatically process the ROM signatures file and retrieve the stored MISR signatures to create the manufacturing patterns. Guidance on where the MISR signature source is obtained, given the existence and validity of ROM content and signature files and settings for the `expected_rom_signature` argument, are provided in [Table 10-11](#) below.

Table 10-11. ROM MISR Signature Source

expected_rom_signature Value	rom_content_file Value	ROM signatures file	MISR Signature Source
<i>binary</i>	N/A	N/A	Use <code>expected_rom_signature</code> value
<i>auto</i>	Specified and readable	N/A	Calculate from <code>rom_content_file</code>
<i>auto</i>	Specified and Not Readable	Readable	Retrieve from ROM signatures file. If no match, results in an error.
		Not readable	Results in an error
<i>auto</i>	Not Specified	Readable	Retrieve from ROM signatures file. If no match, use signature calculated from DftSpecification <code>rom_content_file</code> during MBIST insertion.
		Not readable	Use signature calculated from DftSpecification <code>rom_content_file</code> during MBIST insertion.

- `comparator_id_select : all | none | integer ;`

A property that specifies the individual status of the comparators in the memory interface to be routed to the compare status controller port. Valid values are as follows:

<code>all</code>	Selects the interface's global comparator status signal.
<code>none</code>	Selects a constant logical 1, which implies the comparison result will be forced to consistently pass.

<i>integer</i>	A value between 1 and the number of comparators implemented for this memory. The number of comparators equals the memory word size. The value 1 corresponds to bit 0 of the memory data word.
----------------	---

These usage conditions apply:

- This property is only meaningful when the controller was generated with the [DiagnosisOptions/comparator_selection_mux](#) property set to on in the [DftSpecification](#).

Examples

The following example extracts the MISR register of memory m1 after the test and compares the MISR register content against the hex value 24'h010204:

```
PatternsSpecification(top,rtl,signoff) {
    Patterns(MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep(rom_test) {
            MemoryBist {
                run_mode : run_time_prog ;
                Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
                    DiagnosisOptions {
                        compare_misr : on ;
                        compare_go   : off ;
                    }
                    MemoryInterface(m1) {
                        expected_rom_signature : 24'h010204 ;
                    }
                }
            }
        }
    }
}
```

DramOptions

Provides DRAM AutoRefresh options and properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(pattern_name) {
        TestStep(name) {
            MemoryBist {
                Controller(instance) {
                    DramOptions {
                        run_time_refresh_interval : time | off | auto ;
                        idle_time_refresh_interval : time | off | auto ;
                    }
                }
            }
        }
    }
}
```

Description

The DramOptions wrapper enables you to specify the DRAM AutoRefresh properties that are applied during and after algorithm execution. AutoRefresh can be disabled, or enabled with default or user specified durations between consecutive AutoRefresh operations. AutoRefresh is disabled by default.

The run_time_refresh_interval property specifies the duration between AutoRefresh operations during algorithm execution. The idle_time_refresh_interval property specifies the duration between AutoRefresh operations after algorithm test execution is completed.

Arguments

- run_time_refresh_interval : *time* | *off* | *auto* ;

The run_time_refresh_interval property enables you to specify if AutoRefresh operations are to be automatically performed on a DRAM during the execution of the algorithm. Valid values are:

Table 10-12. DramOptions Parameter Values

<i>time</i>	AutoRefresh is enabled and the real number <i>time</i> specifies the duration between AutoRefresh operations. Valid values for specifying the units for <i>time</i> are as follows: <ul style="list-style-type: none"> • <i>s</i> — specifies the refresh interval in seconds. • <i>ms</i> — specifies the refresh interval in milliseconds. • <i>us</i> — specifies the refresh interval in microseconds. • <i>ns</i> — specifies the refresh interval in nanoseconds. • <i>ps</i> — specifies the refresh interval in picoseconds.
<i>off</i>	The AutoRefresh operations are disabled. This is the default configuration.

Table 10-12. DramOptions Parameter Values (cont.)

auto	AutoRefresh is enabled and the default duration is used between AutoRefresh operations. The default duration is the lowest value of all the RetentionTimeMax properties in the Memory library files divided by the maximum number of memory rows.
------	---

- `idle_time_refresh_interval : time | off | auto ;`

The `idle_time_refresh_interval` property enables you to specify if AutoRefresh operations are to be automatically performed on a DRAM after the test execution completes. Valid parameter values are shown in [Table 10-12](#) above.

Examples

The following example shows a `DramOptions` wrapper that specifies an elapsed time of 250 milliseconds between consecutive AutoRefresh operations for both during and after algorithm execution:

```
PatternsSpecification (top,rtl,signoff) {
    Patterns (MemoryBist_P1) {
        tester_period : 100ns ;
        TestStep (mem_test) {
            MemoryBist {
                Controller (core_inst1) {
                    DramOptions {
                        run_time_refresh_interval : 250ms ;
                        idle_time_refresh_interval : 250ms ;
                    }
                }
            }
        }
    }
}
```

ICLNetworkVerify

Specifies the creation of a structural pattern to verify the proper operation of the IJTAG Network.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        ICLNetworkVerify(id) {
            verify_scan_path : on | off ;
            verify_internal_data_pins : on | off ;
            VerifyOnlyIclModules {
                name_patterns ; // repeatable
            }
            ExcludeIclModules {
                name_patterns ; // repeatable
            }
            VerifyOnlyIclInstances {
                name_patterns ; // repeatable
            }
            ExcludeIclInstances {
                name_patterns ; // repeatable
            }
            ExcludeIclScanRegisters {
                name_patterns ; // repeatable
            }
            ExcludeIclDataPins {
                name_patterns ; // repeatable
            }
        }
    }
}
```

Description

A wrapper that is used to specify the creation of a structural pattern to verify the proper operation of the IJTAG Network. In simulation, the pattern is useful for identifying mismatches between the ICL and Verilog model views. In manufacturing test, it covers manufacturing defects that can exist in the IJTAG network circuitry.

When the property verify_scan_path is on, the patterns are created by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictable captured values of the ScanRegisters are compared and the scan registers are loaded with all 0s with a single 1 closest to ScanIn. The ScanRegister is then parked in pause state and then unloaded to make sure the same data comes out, making sure the ScanRegisters are actually the length documented in the ICL model.

When verify_internal_data_pins is on, the internal DataOutPorts are forced and those values observed using ScanRegisters unloading. The internal DataInPorts are controlled using ScanRegisters loading and the resulting value directly observed on the internal DataInPorts. This feature is only usable in simulation and is useful for finding mismatches between the circuits and their ICL descriptions.

Arguments

- `verify_scan_path : on | off ;`

A property that specifies whether the patterns is to verify the integrity of the scan paths. When the property is on, the patterns are created by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictable captured values of the ScanRegisters are compared and the scan registers are loaded with all 0s with a single 1 closest to ScanIn. The ScanRegister is then parked in pause state and then unloaded to make sure the same data comes out, making sure the ScanRegisters are actually the length documented in the ICL model. The opposite pair of patterns are also applied as shown here:

```
load 100000...000 expect captures values described in ICL
load 000000...000 expect 100000...000
load 011111...111 expect captures values described in ICL
load 111111...111 expect 011111...111
```

- `verify_internal_data_pins : on | off ;`

A property that specifies whether the patterns is to verify that data can be properly controlled at the internal DataInPorts and observed from the internal DataOutPorts. When the property is on, the internal DataOutPorts are forced and those values observed using ScanRegisters unloading. The values on internal DataInPorts are controlled using ScanRegisters loading and the resulting value directly observed on the internal DataInPorts. This feature is only usable in simulation and is useful for finding mismatches between the circuits and their ICL descriptions.

Note that ICL Extraction sets the `tessent_design_instance` attribute to the pre-synthesis instance names whenever those names are available. During the creation of the data pin verification patterns, the actual design instance names are ignored. The verification pattern generator only uses the `tessent_design_instance` attribute to obtain the references to the design pins. Consequently, the rtl design files provided to ICL extraction must be used during the simulation of the verification patterns, at least in case that the synthesis modifies the design hierarchy (for example, because of “generate” loops). In order to obtain data pin verification patterns which can be simulated using the synthesized netlist with instance names that have been modified by synthesis, you either have to modify the `tessent_design_instance` attributes accordingly or run ICL Extraction based on the synthesized netlist from the beginning (without providing rtl files to the tool).

- `VerifyOnlyIclModules/name_patterns ;`

A Data wrapper used to specify that the test should only focus on a set of ICL modules. The `name_patterns` are glob patterns matching ICL module names instantiated below the current ICL design.

- `ExcludeIclModules/name_patterns ;`

A Data wrapper used to specify that the test should exclude a set of ICL modules. The `name_patterns` are glob patterns matching ICL module names instantiated below the current ICL design. ICL modules having the `tessent_ignore_during_icl_verification` (see [icl_module](#)) attribute set to on are automatically excluded. Use this attribute on ICL

modules which are not truly IJTAG compliant and cannot pass the structural test performed in this test. One such example is the Tesson memory BIST controller. Because of its asynchronous interface, its BIST_CLOCK must be toggling at least 4 time faster than TCK to enable proper shift operation. This dependency on the ClockPort is not documented in ICL such that ICLNetworkVerify does not know to enable the source of the clock port. Such modules are tested as part of the memory BIST operations. They are also tested during scan test as the memory BIST controllers are scan testable.

- VerifyOnlyIclInstances/*name_patterns* ;

A Data wrapper used to specify that the test should only be performed on set of ICL instances. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design.

- ExcludeIclInstances/*name_patterns* ;

A Data wrapper used to specify that the test should exclude a set of ICL instances. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design. ICL instances having the tessent_ignore_during_icl_verification attribute set to on are automatically excluded.

- ExcludeIclScanRegisters/*name_patterns* ;

A Data wrapper used to specify that the test should exclude a set of ICL Scan Register. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design. ICL Scan Registers having the tessent_ignore_during_icl_verification attribute set to on are automatically excluded. This wrapper is only used when verify_scan_path is set to on.

- ExcludeIclDataPins/*name_patterns* ;

A Data wrapper used to specify that the test should exclude a set of ICL DataPorts. The *name_patterns* are glob patterns matching ICL DataPorts found below the current ICL design. This wrapper is only used when the verify_internal_data_pins option is set to on. DataInPorts and DataOutPorts that have the tessent_ignore_during_icl_verification attribute set to on in their ICL definition are automatically excluded. This switch is only used when the [create_icl_verification_patterns -data_pin_test](#) option is set to on.

Examples

Example 1

The following example specifies the creation of the ICL network verification patterns set. The ExcludedIclModules wrapper was used to exclude all ICL Module matching abc*.

Alternatively, the attribute tessent_ignore_during_icl_verification could have been set to true in the ICL module to exclude it automatically.

```
ICLNetworkVerify {
    ExcludedIclModules {
        abc* ;
    }
}
```

Example 2

The following example demonstrates creating one ICLNetworkVerify step using the signoff for the sub-physical regions and using the to replace what is being done in the default manufacturing pattern specification generated.

```
Patterns(ICLNetwork) {
    ICLNetworkVerify(top) {
        // Testing ICL scan registers outside all physical blocks
        ExcludeIclInstances {
            m8051_wrap_inst_m8051_inst1;
            m8051_wrap_inst_m8051_inst2;
            piccpu_inst1;
        }
    }
    ICLNetworkVerify(top_1) {
        // Testing ICL scan registers,
        // including those in physical block ICL instance
        // 'm8051_wrap_inst_m8051_inst1'
        ExcludeIclInstances {
            m8051_wrap_inst_m8051_inst2;
            piccpu_inst1;
        }
    }
    ICLNetworkVerify(top_2) {
        // Testing ICL scan registers,
        // including those in physical block ICL instance
        // 'm8051_wrap_inst_m8051_inst2'
        ExcludeIclInstances {
            m8051_wrap_inst_m8051_inst1;
            piccpu_inst1;
        }
    }
    ICLNetworkVerify(top_3) {
        // Testing ICL scan registers,
        // including those in physical block ICL instance 'piccpu_inst1'
        ExcludeIclInstances {
            m8051_wrap_inst_m8051_inst1;
            m8051_wrap_inst_m8051_inst2;
        }
    }
}
```

As shown, the ICLNetworkVerify(`top`) excludes all sub-physical blocks instances. Then, in turn, each sub-physical block instance is *not* excluded in subsequent ICLNetworkVerify(`<design_name>_<n>`) wrappers.

A comment is added outside each ExcludedInstances wrapper that explains the objective of this particular ICLNetworkVerify wrapper.

Related Topics

[create_icl_verification_patterns](#)

ProcedureStep

Inserts arbitrary procedure steps into the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    Patterns(patterns_name) {
        ProcedureStep(procedure_step_name) {
            svf_file : file_path ;
            maintain_model_state_from_svf : on | off ;
            run_before_dft_control_settings : on | off ;
            iCall(iproc_name_path) { // repeatable
                iProcArguments {
                    argument_name : argument_value ; // repeatable
                }
            }
            wait_time : time ;
            tester_cycles : integer ;
            tck_cycles : integer ;
            AdvancedOptions {
                force_voltage(pin_name) : voltage ; // repeatable
                split_patterns_file : on | off ;
                network_end_state : keep | reset | initial ;
            }
        }
    }
}
```

Description

The ProcedureStep wrapper is used to insert arbitrary procedure steps into the pattern set. The events of the procedure step consist of a test sequence expressed as a SVF (serial vector format) file or as an iProc followed by a pause of arbitrary length.

The example that follows is an ICL module that describes a programmable clock divider. If this ICL was visible when the [process_dft_specification](#) command was issued, a TDR was inserted to provide controllability of the DataInPorts from the IJTAG network. The ICL module defines the relationship between the clk_out and clk_in as well as the default division ratio between them. The DataInPort rst can be used to reset the divider and the DataInPort ratio[1:0] can be used to program the division ratio. As described in the Description section of the [add_clocks](#) command, an add_clocks command will automatically be inferred when elaborating the design in the DFT context.

Notice how the “rst” ICL port has the [tessent_use_in_dft_specification](#) attribute set to “auto_no_pi”. When unspecified, this attribute defaults to “auto”. With “auto”, the input pin will be intercepted or sourced by a DataOutPort of a TDR if the port is seen to not already be sourced by a valid DataOutPort or a primary input port. When you set it to “auto_no_pi”, the pin will be equipped with a local IJTAG control even if it is seen to already be sourced by a primary input. You want to use the “auto_no_pi” value for ports like this “rst” port because they are often sourced by a system level reset port. Even though it is possible to toggle the system reset to reset the clock divider, the effect of the reset pulse will be felt by the entire system and you

may affect already running parts of the circuit. When you provide local controllability, it becomes possible to reset the clock divider without affecting the rest of the chip. See the description of [create_dft_specification](#) for more information about how DataInPorts are handled during DFT insertion.

If you add the [tessent_default_procedure_name](#) and optionally the [tessent_default_procedure_parameters](#) in the ToClockPort as shown below, the [create_patterns_specification](#) will automatically add the ProcedureStep to iCall the procedure with the specified parameters if it sees the clock of the memoryBIST controller going through this ToClockPort. It is critical that all elements you specify in the ToClockPort be consistent. The default proc and parameters must program the ToClockPort to behave as its Source, FreqDivider and FreqMultiplier properties indicates.

If the [create_patterns_specification](#) command sees that multiple cascaded modules need to be initialized, it will initialize them in separate ProcedureStep such that the element feeding the other is initialized first. See [Example 2](#) where a PLL is initialized first followed by the clock divider in its fanout.

```
Module clk_div {
    Attribute keep_active_during_scan_test = "true" ;
    ClockPort clk_in ;
    ToClockPort clk_out {
        Attribute tessent_default_procedure_name = "init" ;
        Attribute tessent_default_procedure_parameters = "ratio 4" ;
        Source clk_in ;
        FreqDivider 4 ;
    }
    DataInPort rst {
        //This attribute makes sure the pin will be intercepted by
        //create_dft_specification even if the pin is sourced by a PI.
        //Because the rst pin is often shared with many other things
        //in the chip, the local interception makes sure the clock divider can
        //be reset by its iProc without affecting the rest of the design.
        Attribute tessent_use_in_dft_specification = "auto_no_pi";
    }
    DataInPort ratio[1:0] ;
}
```

The basic PDL iProcs to program the divider is shown here:

```
iProcsForModule clk_div
iProc init {{ratio 4}} {
    iClock clk_in
    iWrite rst 0b1
    iWrite ratio [expr int(log($ratio)/log(2))]
    iApply
    iWrite rst 0b0
    iApply
    iRunLoop 8 -sck clk_in
    iClockOverride clk_out -freqDivider $ratio
}
```

The iProc accepts a ratio argument that defaults to 4, and uses this value to program the divider. Once the divider is programmed, the [iClockOverride](#) command is issued to reflect that the division ratio has been changed. The later [iClock](#) commands that trace to this ToClockPort will see the overridden value. The basic PDL iProc shown above is correct in terms of the PDL commands but lacks the Tcl code needed to validate the ratio argument.

The enhanced procedure shown below has the proper validation code. The code at the top validates the ratio value and gives an error message when incorrect. When the iProc is called within a ProcedureStep wrapper, the iProc will be called with a parameter that defines the validation_dict argument containing an entry for each parameter. In the example below, the entry “iproc_parameters ratio” is defined and contains the name of the property inside the ProcedureStep that defines its value. You can use this property to attach the error message to the property using the [add_config_message](#) command as shown below.

Finally, the validation_dict argument contains an entry called “validate_only”. It is expected that the code “if {[dict get \$validation_dict validate_only]} {return}” exists in the iProc such that the iProc can be called with this entry set to 1 during the validation step of the [process_patterns_specification](#) command in order to run the validation code without actually running the PDL command. When the argument called “validation_dict” is not present, the iProc is not run when you call the [process_patterns_specification](#) command with the -validate_only option. This means that incorrect values of arguments passed to iProcs from a ProcedureStep wrapper will only be detected when you run the [process_patterns_specification](#) command without the -validate_only option and the error message will only show up in the transcript and won’t be attached to the property objects in the GUI.

```
iProcsForModule clk_div
iProc init {{ratio 4} {validation_dict {validate_only 0}}} {
    ##### Validation part #####
    if {$ratio ni {1 2 4 8}} {
        set err "Illegal value '$ratio' for option 'ratio'. Must be 1, 2, 4, or
8. Default is 4."
        if {[dict exists $validation_dict iproc_parameters ratio]} {
            add_config_message -display $err \
                -config [dict get $validation_dict iproc_parameters ratio]
        } else {
            display_message $err
        }
        return -code error
    }
    if {[dict get $validation_dict validate_only]} {
        # Skip PDL part
        return
    } ##### PDL part #####
    iClock clk_in
    iWrite rst 0b1
    iWrite ratio [expr int(log($ratio)/log(2))]
    iApply
    iWrite rst 0b0
    iApply
    iRunLoop 8 -sck clk_in
    iClockOverride clk_out -freqDivider $ratio
}
```

Arguments

- *procedure_step_name*

A string that uniquely identifies the step within a pattern set. All of the procedure_step_name and test_step_name names must be unique within a Patterns wrapper. The string is only allowed to contains letters, numbers, and underscores.

- *svf_file :file_path ;*

A property that specifies a file that defines an SVF sequence to be included in the ProcedureStep. The svf_file and iCall(iproc_name_path) properties are mutually exclusive.

There are limitations on what can be in the svf_file. See [import_patterns_from_svf](#) command for a complete list. If the svf_file includes a reset of the Mentor Graphics TAP, then the Mentor Graphics TAP must be left in the Run-Test-Idle state (at the end of the svf_file). An example of this use model is if the svf_file contents must be run to provide access to the Mentor Graphics TAP.

The file_path must have one of the following formats:

- A simple file name with no directory path. In this case, the file must exist beside the ICL file.
- A null string. In this case, no procfile is used.
- An absolute path starting with a “/”.

- A relative path starting with “./” or “../”. In this case, the file must exist relative to the current working directory.
- `maintain_model_state_from_svf : on | off ;`
An optional property that specifies how the SVF reader operates. When set to off, the reader operates in “blackbox mode”, in that it will not apply the SDR (Scan Data Register) and SIR (Scan Instruction Register) commands to the internal state of the IJTAG network, nor will it check if the internal state of the IJTAG network matches the length of the provided SDR/SIR data. The default value is on.
- `run_before_dft_control_settings : on | off ;`
The ProcedureStep/TestStep steps are always relative to each other in the order in which they appear in the PatternsWrapper. The settings described in the DftControlSettings wrappers are always applied first no matter where the wrapper is positioned.
In some situations, you may need to run one or more custom ProcedureStep before the DftControlSettings can be applied, for example powering up a section of the device. Specifying the `run_before_dft_control_settings` on the N first ProcedureStep wrapper will run them in their current relative order before the DftControlSettings are applied.
An error is generated if you have a TestStep or a ProcedureStep with `run_before_dft_controling_settings` set to off above a ProcedureStep with `run_before_dft_controling_settings` set to on.
- `iCall(iproc_name_path)`

A Wrapper that specifies the hierarchical ICL path to an iProc to be called. For example, an iProc named “reset” which has been registered against an ICL module mod1 that is instantiated as “mod1_inst” in the ICL is referenced as `iCall(mod1_inst.reset)`. The iCall wrapper and svf_file properties are mutually exclusive.

When many `iCall(iproc_name_path)` wrappers exists in a ProcedureStep wrapper, the associated iCalls are processed in an ‘iMerge -begin ... iMerge -end’ block so that their operations are performed in parallel. When there is only one iCall in the wrapper, it is not processed within an iMerge block. If the iProcs you want to iCall within a ProcedureStep wrapper contains instructions that are incompatible with the merging procedure (for example, `iApply -end_in_pause`), make sure to include it by itself inside one ProcedureStep wrapper to avoid it being process in an iMerge block.

The iProc for the ICL module must have been sourced into the tool prior to executing the [process_patterns_specification](#) command; otherwise, an error will be generated.

Note

 If your store your iProcs in a file called XXX.pdl and an ICL module was read in from a file called XXX.icl and the two files existed in the same directory, the content of the file XXX.pdl will be collected during ICL extraction and stored into a file called <design_name>. pdl inside the TSDB. This file will be auto loaded when ICL is elaborated, removing the need for you to manually source the iProcs before they can be referenced by the iCall() wrapper.

- `iCall(iproc_name_path)/iProcArguments/argument_name : argument_value ;`

A Data wrapper that specifies argument-values pairs that specifies one or more arguments to pass to the iProc identified with the `iproc_name` property. The `argument_name` must exactly match the name of an argument of the actual iProc. Tesson Shell generates an error if the argument does not exist. The order of the arguments is not important as long as they formally match those defined for the iProc.

The `argument_value` is automatically passed to the iProc in the right position.

Tesson Shell generates an error if a required argument of the iProc is missing from the `iProcArguments` wrapper.

If an optional argument is not specified, the default value of that argument is passed to the iProc when called.

Consider the following example containing the declaration of an iProc called “reset”:

```
iProc reset { initial_delay {final_delay 10us} } {...}
```

The following configuration data:

```
iCall(mod1_inst.reset) {
    iProcArguments {
        initial_delay : 1us;
    }
}
```

results in the following call to the iProc because the value of `final_delay` is optional and defaults to 10 us:

```
iCall mod1_inst.reset 1us 10us
```

The following configuration data results in an error because the `initial_delay` argument is required:

```
iCall(mod1_inst.reset) {
    iProcArguments {
        final_delay : 22us;
    }
}
```

If you have an iProc that uses “args” as an argument, and you want to pass a series of option-value pairs to the args option, you use the following syntax:

```
iProcArguments {
    args : "option1 value1
            option2 value2
            optionN valueN";
}
```

- `wait_time : time ;`

A property that specifies a wait loop at the end of the procedure step. The `wait_time` property is mutually exclusive to the `tester_cycles` and `tck_cycle` properties. It expresses a

wait loop in term of times. The specified time value is divided by the tester_period value and the ceiling value is used to create a wait loop.

- tester_cycles : *integer* ;

A property that specifies a wait loop at the end of the procedure step. The tester_cycles property is mutually exclusive to the wait_time and tck_cycle properties. It expresses a wait loop in term of tester_cycles. The specified integer value is used to create a wait loop.

- tck_cycles : *integer* ;

A property that specifies a wait loop at the end of the procedure step. The tck_cycles property is mutually exclusive to the wait_time and tester_cycles properties. The specified integer value is multiplied by the tck_ratio value to create a wait loop.

- AdvancedOptions/force_voltage(*pin_name*) : *voltage* ;

A repeatable property that specifies that the current test step needs a special voltage to be applied on a specific port of the device. This property is typically used in TestStep when MemoryBisr is used to program the fusebox. The property does not actually force the voltage. It only generates a special annotation near the top of the patterns file to let the test engineer know that test program commands are needed to supply the needed voltage on the given port. As soon as a single force_voltage property is specified, the split_patterns_file : on; property is implied to allow the insertion of the test program commands needed to apply the voltage values.

Here is an example of the special annotation inside a stil file:

```
Ann {* Before applying this pattern, the following actions are
      required: *}
Ann {*} - Apply +0.25V on port PA {}}
```

```
Ann {*} - Apply -0.25V on port PB {}}
```

- AdvancedOptions/split_patterns_file : on | off ;

A property that splits the pattern set into multiple files. The file containing the previous patterns is closed and a new one is created for the current step. The file name is the name of the Patterns wrapper initially. When a step with split_patterns_file set to on is reached, the new file name consists of the name of the Patterns wrapper concatenated with an underscore to the name of the TestStep wrapper as shown here:

```
<pattern_name>_<test_step_name>
```

The split patterns file contains a special annotation near the top of the file that reads like this:

```
Ann {*} This pattern file requires some actions taken on the ATE. *
Ann {*} This pattern assumes it is following another one: *
Ann {*} the power and clocks must be kept alive between *
```

```
Ann {*} the two patterns and all pins kept their state. {}}
```

- AdvancedOptions/network_end_state : `keep` | `reset` | `initial` ;

A property that specifies the action to take at the end of the ProcedureStep. The options for this property are named the same and perform the identical functions as the `close_pattern_set` command's arguments. For details, see the [close_pattern_set](#) command.

Examples

Example 1

The following example shows a Patterns wrapper containing a single ProcedureStep that consists of an initial pause of 4 milliseconds followed by the call of an iProc called “run” on the ICL instance “core1_i1.instrumenta_i1”. The parameter mode is passed to the iProc with a value equal to “random”.

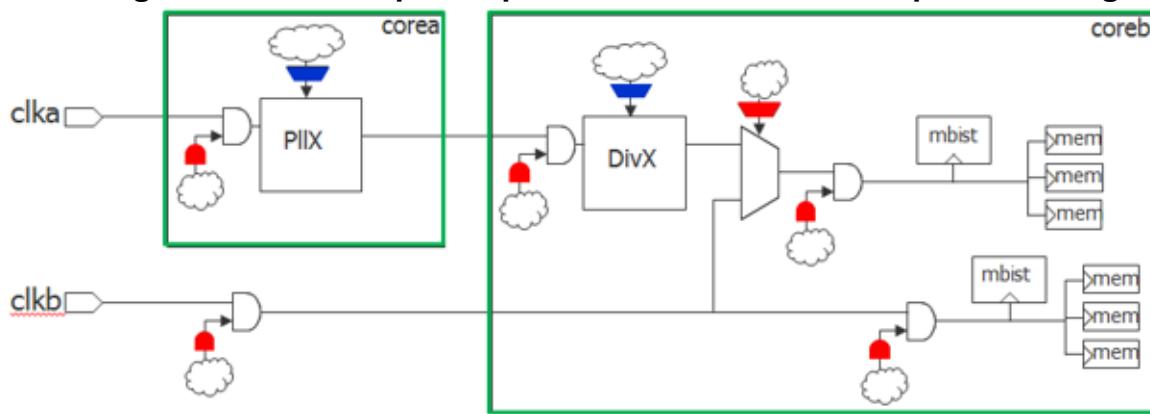
```
PatternsSpecification(mychip, gate, signoff) {
    Patterns(P1) {
        ProcedureStep(my_icall) {
            wait_time : 4ms ;
            iCall(core1_i1.instrumenta_i1.run) {           iProcArguments {
                mode : random ;
            }
        }
    }
}
```

Example 2

The following example shows how the Patterns wrapper was auto created by the [create_patterns_specification](#) command to perform signoff simulation for the memory BIST controller found inside coreb (See [Figure 10](#)). Notice how the content of the DftControlSettings, the SimulationOptions/LowerPhysicalBlockInstances and the ProcedureSteps wrappers were created. The command traces all the clocks of the MemoryBist controllers found in the TestStep/MemoryBist wrapper. For every physical block entered, it asserts the `dft_control_enable`, `nonscan_test`, and `all_test` dft control signals it finds in them. For more information about those signals, see the [add_dft_control_points](#), [add_dft_clock_enables](#), and [add_dft_clock_mux](#) command descriptions. Normally, the signoff simulation of a given physical block instance is done using the interface view for all other physical block instances. In this example, the clock traced through the corea_i1 instance so the full view of corea_i1 is requested when simulating coreb_i1 in order for it to get its clocks. Finally, notice how two ProcedureStep wrapper were auto inserted to initialize the PLL and clock divider seen when tracing the clocks. The clock divider is initialized after the PLL because it requires the output of the PLL to be running before it is initialized. This automatic inferring of the ProcedureStep only happens if you have defined the [tessent_default_procedure_name](#) attribute in the ToClockPort of the ICL model for the PLL and clock divider as described above in the Description section.

```
PatternsSpecification(chip,rtl,signoff) {
    Patterns(MemoryBist_P2) {
        tester_period : 100ns ;
        DftControlSettings {
            corea_i1.all_test : 1 ;
            corea_i1.nonscan_test : 1 ;
            coreb_i1.all_test : 1 ;
            coreb_i1.nonscan_test : 1 ;
            all_test : 1 ;
            nonscan_test : 1 ;
        }
        ClockPeriods {
            clka : 3.00ns;
            clkb : 9.50ns;
        }
        SimulationOptions {
            LowerPhysicalBlockInstances {
                corea_i1 : full ;
                coreb_i1 : full ;
            }
        }
    }
    ProcedureStep(clock_setup1) {
        iCall(corea_i1 pll.init) {
            iProcArguments {
                multiply : 10 ;
            }
        }
    }
    ProcedureStep(clock_setup2) {
        iCall(coreb_i1 clk_div.init) {
            iProcArguments {
                ratio : 4 ;
            }
        }
    }
    TestStep(run_time_prog) {
        MemoryBist {
            ...
        }
    }
}
```

Figure 10-60. Example Chip with Two Cores and Complex Clocking



Related Topics

[create_patterns_specification](#)
[process_patterns_specification](#)

EnableGroupInfo

Creates groups of Boundary Scan enable cells that can then be referenced using the BoundaryScan/RunTest/enable_group_info property.

Usage

```
PatternsSpecification(design_name ,design_id,pattern_id) {
    EnableGroupInfo(enable_group_info_id) {
        enable_cell_numbers : cell_number, ...;
    }
}
```

Description

The EnableGroupInfo wrapper is used to group enable cells together that will be tested simultaneously. The enable_group_info_id property can be used in the enable_group_info property to specify the enable groups to be tested separately and the sequence in which the enable groups are tested.

Arguments

- *enable_group_info_id*

A string that uniquely identifies the EnableGroupInfo wrapper such that it can be referenced using the BoundaryScan/enable_group_info property. All enable cells that are not part of the listed enable groups, whether they are part of a not listed enable group or part of a no enable group, are added to a new artificial group and all tested simultaneously.

- *enable_cell_numbers : cell_number, ...;*

A property used to list the enable Boundary Scan cell numbers in the design to be tested simultaneously. The cell number corresponds to the cell number listed in the BSDL description.

Examples

The following example defines two EnableGroupInfo wrappers with different enable cells listed in each. The BoundaryScan wrapper shows two BoundaryScan wrappers referencing each enable group.

```
PatternsSpecification(mychip,final,manufacturing) {
    Patterns(bscan) {
        TestStep(TS1) {
            BoundaryScan {
                enable_group_info : g1;
                RunTest(output) {
                }
            }
        }
        TestStep(TS2) {
            BoundaryScan {
                enable_group_info : g2;
                RunTest(output) {
                }
            }
        }
    }
    EnableGroupInfo(g1) {
        enable_cell_numbers : 3, 34, 78, 89;
    }
    EnableGroupInfo(g2) {
        enable_cell_numbers : 11, 37, 57, 101;
    }
}
```

SdfInfo

Defines SDF file usage scenarios that can be referenced by the Patterns/SimulationOptions/`sdf_info` properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    SdfInfo(sdf_info_id) {
        SdfFile(path_to_file) {
            instance_names      : instance_name, ... ;
            include_macro_name : macro_name ;
            exclude_macro_name : macro_name ;
        }
    }
}
```

Description

The SdfInfo wrapper is used to defined SDF file usage scenarios that can be referenced by the [Patterns/SimulationOptions](#)/`sdf_info` properties.

The various SDF files will be annotated on the specified instance path or the top-level module when the `instance_names` property is left empty. You can request the `sdf` annotation commands to only run or to not run when a given macro name is defined on the simulator command line.

Arguments

- `path_to_file`

A string that refers to an SDF file. Relative path are expressed relative to the current working directory; they are converted to an absolute path in the test bench such that the simulation can be performed in any directory.

- `instance_names : instance_name, ... ;`

A property that defines one or many Verilog instance paths that the SDF file applies to. When the property is specified as a null string (the default value), the SDF file is applied to the current design. The instance path names uses the period “.” because the hierarchy separator is used in a Verilog simulator.

- `include_macro_name : macro_name ;`

A property that defines a macro name that will be used to surround the `sdf_annotation` command with a ‘`ifdef macro_name`’ statement. When you specify this property, the `sdf` annotation only happens if you pass “`+define+macro_name`” on the simulator command line. The `sdf_annotation` command is not surrounded by a ‘`ifdef`’ statement unless this property is specified. This feature is very useful when you want to simulate with more than one version of an SDF for the same test benches as shown in the following example.

- `exclude_macro_name : macro_name ;`

A property used to define a macro name that will be used to surround the `sdf_annotation` command with a ‘`ifndef macro_name`’ statement. When you specify this property, the `sdf`

annotation can be suppressed by passing “+define+macro_name” on the simulator command line. The sdf_annotation command is not surrounded by a ‘ifndef statement unless this property is specified.

Examples

The following example defines an SdfInfo wrapper that loads an SDF file called “top.sdf_slow” for the current design and another SDF file called “corea.sdf_slow” for two instances of the physical block “corea” when the +define+sdf_slow macro is passed to the simulator. It loads an SDF file called “top.sdf_fast” for the current design and another SDF file called “corea.sdf_fast” for two instances of the physical block “corea” when the +define+sdf_fast macro is passed to the simulator.

```
SdfInfo(chip_corea) {
    SdfFile(..../SDFFiles/top.sdf_slow) {
        include_macro_name : sdf_slow ;
    }
    SdfFile(..../SDFFiles/corea.sdf_slow) {
        instance_names : tlb.corea_i1, tlb.corea_i2 ;
        include_macro_name : sdf_slow ;
    }
    SdfFile(..../SDFFiles/top.sdf_fast) {
        include_macro_name : sdf_fast ;
    }
    SdfFile(..../SDFFiles/corea.sdf_fast) {
        instance_names : tlb.corea_i1, tlb.corea_i2 ;
        include_macro_name : sdf_fast ;
    }
}
```

Below is the corresponding Verilog module that is created and simulated in parallel to the chip to load the SDF files.

```
module P1_sdf
initial begin
  `ifdef sdf_slow
    sdf_annotate(/home/ver_eng/SDFFiles/top.sdf_slow,
                TB.top_inst);
  `endif
  `ifdef sdf_slow
    $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_slow,
                  TB.top_inst.tlb.corea_i1);
    $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_slow,
                  TB.top_inst.tlb.corea_i2);
  `endif
  `ifdef sdf_fast
    sdf_annotate(/home/ver_eng/SDFFiles/top.sdf_fast,
                 TB.top_inst);
  `endif
  `ifdef sdf_fast
    $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_fast,
                  TB.top_inst.tlb.corea_i1);
    $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_fast,
                  TB.top_inst.tlb.corea_i2);
  `endif
end
endmodule
```

LoadBoardInfo

A wrapper used to define the different tester and load board capabilities.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
    LoadBoardInfo(load_board_info_id) {
        dot6_ttest : ttest_time ;           // default: auto
        TesterInterface {
        }
        Loopbacks {
        }
        ACLoopbacks {
        }
    }
}
```

Description

The LoadBoardInfo wrapper is used to define the different tester and load board capabilities and configurations. It can be referenced using the [PatternsSpecification/Patterns/load_board_info](#) property—see “[Patterns](#)” on page 3576.

You would need this wrapper specified and referenced when:

- Your tester supports applying a 'Z' pattern state (highZ) to an input-only pin, and you want to use that capability in your BoundaryScan/[RunTest](#) dot6_ac_input test pattern.
- Your load board features loopbacks between output and input ports used in patterns, including boundary scan tested ports for [BoundaryScan](#) patterns.
- On your load board, some of your tested ports:
 - Have pull resistors.
 - Are uncontrollable by the tester (input and bidirectional ports).
 - Are unobservable by the tester (output and bidirectional ports).

When no LoadBoardInfo is specified, Tesson Shell assumes:

- All input and bidirectional tested ports are controllable.
- The tester cannot disable the drive of an input-only port.
- All output and bidirectional ports can be observed for states 0, 1 and Z.

Arguments

- *load_board_info_id*

A string that uniquely identifies the LoadBoardInfo wrapper such that it can be referenced using the [PatternsSpecification/Patterns/load_board_info](#) property.

- `dot6_ttest : ttest_time ;`

An optional time value that specifies the minimum time required by the slowest coupling capacitor, among all AC Loopbacks, to fully discharge (under 1% of charge). The value of this property (also referred to as TTest in the IEEE 1149.6 standard) is used mainly by DC-related BoundaryScan tests such as `dot6_dc_input` and `dot6_dc_output`. They ensure that DC levels are systematically blocked by all coupling capacitors and therefore cannot be captured by the test receiver.

This property's value defaults to three times the slowest time constant along all AC pads that have an on-chip high-pass or low-pass filter as recommended by the IEEE 1149.6 standard. If no on-chip filter is present or if no BSDL file can be found, then the value defaults to three periods of TCK.

Examples

The following example will be described in the next sections.

```
LoadBoardInfo(tester1) {
    TesterInterface {
        Port (TRST) {
            control      : none;
            observation : none;
        }
        Port (IN1) {}
        Port (IN_DV1) {
            differential_inverse_of : IN_DV1_N; //only specify if no BSDL
        }
        Port (IN_AC1) {}
        Port (IN_AC1_DV) {
            differential_inverse_of : IN_AC1_DV_N; //only specify if no BSDL
        }
    }
    Loopbacks {
        IN1 : OUT1;
        IN_DV1 : OUT_DV1;
    }
    dot6_ttest : 15.0e-09s;
    ACLoopbacks {
        IN_AC1 : OUT_AC1;
        IN_AC1_DV : OUT_AC1_DV;
    }
}
```

TesterInterface

The TesterInterface wrapper enables you to specify which design ports are to be contacted to a tester channel

Usage

```
LoadBoardInfo(load_board_info_id) {
    TesterInterface {
        default_control      : auto | none | two_state_weak |
        two_state_strong |                               three_state | tied_high | tied_low ;
        default_observation : auto | none | two_state | three_state ;
        default_pull_resistor : from_bsdl | none | up | down ;
        Port(port_name) {
            control          : from_default | none | two_state_weak |
            two_state_strong | three_state | tied_high | tied_low ;
            observation       : from_default | none | two_state |
            three_state ;
            pull_resistor     : from_default | from_bsdl | none |
            up | down ;
            differential_inverse_of : port_name ;
        }
    }
}
```

Description

The TesterInterface wrapper enables you to specify which design ports are to be contacted to a tester channel during test pattern execution and their capabilities. Specifying this wrapper will apply the default_control, default_observation and default_pull_resistor values to all ports of the current design that are not specified in a Port wrapper.

Arguments

- default_control : auto | none | two_state_weak | two_state_strong | three_state | tied_high | tied_low ;

This property defines the control capabilities of the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the control : from_default value specified. The default value is “auto”, which resolves to “two_state_strong” for input ports, “none” for output ports and “three_state” for inout ports.

- default_observation : auto | none | two_state | three_state ;

This property defines the observation capabilities of the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the observation : from_default value specified. The default value is “auto”, which resolves to “three_state” for output and inout ports and to “none” for input ports.

- default_pull_resistor : from_bsdl | none | up | down ;

The property defines the polarity of the external weak pull resistor set by the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the

`pull_resistor` : `from_default` value specified. The default value is “`from_bsdl`”. For more details, see the property description of Port/`pull_resistor`.

- `Port(port_name)`

This identifier defines which port the following properties apply to. This must be an existing port of the current design.

When specifying a differential pair, you must only specify the positive leg’s name. If you have Tessent Boundary Scan DFT inserted, then the negative leg will be inferred from the BSDL file, if not, then you should use the `differential_inverse_of` property to specify the name of the negative leg of this differential port.

- `Port/control : from_default | none | two_state_weak | two_state_strong | three_state | tied_high | tied_low ;`

This property defines the control capabilities of the tester on the identified port. The default is “`from_default`”, which specifies the control capabilities will be defined by the `default_control` property. For an input-only port, setting the value “`none`” means the port is not contacted by the tester and will not be part of Tessent Boundary Scan patterns.

- `Port/observation : from_default | none | two_state | three_state ;`

This property defines the observation capabilities of the tester on the identified port. The default is “`from_default`”, which specifies the observation capabilities will be defined by the `default_observation` property. For an output-only port, setting the value “`none`” means the port is not contacted by the tester and will not be part of Tessent Boundary Scan patterns.

- `Port/pull_resistor : from_default | from_bsdl | none | up | down ;`

The property defines the polarity of the external weak pull resistor set by the tester on the port if any. The default value is “`from_default`”, which specifies the polarity will be defined by the `default_pull_resistor` property. When the value is “`from_bsdl`”, Tessent Shell will try to infer the value from the disable result value in the `BOUNDARY_REGISTER` section of the BSDL file if one can be found. A disable result value of “`weak0`” will infer this property’s value to “`down`” and a disable result value of “`weak1`” will infer this property’s value to “`up`”.

- `Port/differential_inverse_of : port_name ;`

This property is used to specify the negative leg of a differential port pair that complements this positive leg of the port. You should only use this property if you do not have Tessent BoundaryScan DFT.

Examples

In the example, the tester is not contacting the TRST TAP port and 4 inputs. IN_DV1 is a differential input so the control, observation and `pull_resistor` properties will be applied automatically to complimentary port IN_DV1_N.

Note: If a TAP TRST port is listed as uncontrollable, it will be specially treated. Tessent Shell will infer a CT1 constraint on that port.

```
LoadBoardInfo(tester1) {
    TesterInterface {
        Port (TRST) {
            control : none ;
            observation : none ;
        }
        Port (IN1) {}
        Port (IN_DV1) {
            differential_inverse_of : IN_DV1_N ; //only specify if no BSDL
        }
        Port (IN_AC1) {}
        Port (IN_AC1_DV) {
            differential_inverse_of : IN_AC1_DV_N ; //only specify if no BSDL
        }
    }
    [...]
}
```

Loopbacks

The Loopbacks wrapper specifies one or more connection loopbacks from source ports to destination ports.

Usage

```
LoadBoardInfo(load_board_info_id) {
    Loopbacks {
        destination_port : source_port ; // repeatable
    }
}
```

Description

The Loopbacks wrapper specifies one or more connection loopbacks from source ports to destination ports of the current design. Loopbacks specify direct connections, as opposed to ACLoopbacks, which connect source ports to destination ports through a coupling capacitor.

Usage Conditions

Note the following:

- *source_port* and *destination_port* must be output and input direction respectively. Inout ports are not allowed.
- An 1149.1 source must loop back to an 1149.1 destination and an 1149.6 source must loop back to an 1149.6 destination. Interconnecting 1149.1 and 1149.6 ports together is not allowed.
- To loop back differential pairs, you must only specify the positive leg's name. The negative leg will be inferred from the BSDL file or taken from the *differential_inverse_of* properties of the ports.

Arguments

- *destination_port : source_port* ;

This port pair describes a direct connection that is to be made in the simulation test bench and assumed by the test patterns.

Examples

In the example, OUT1 and OUT_DV1 are looping back to uncontacted ports IN1 and IN_DV1 respectively. Because IN_DV1 and OUT_DV1 are differential ports, their negative leg will also automatically be looped back.

```
LoadBoardInfo(tester1) {
    [...]
    Loopbacks {
        IN1 : OUT1 ;
        IN_DV1 : OUT_DV1 ;
    }
    [...]
}
```

ACLoopbacks

The ACLoopbacks wrapper specifies one or more connection loopbacks from AC source ports to AC destination ports.

Usage

```
LoadBoardInfo(load_board_info_id) {
    ACLoopbacks {
        destination_port : source_port ; // repeatable
    }
}
```

Description

The ACLoopbacks wrapper specifies one or more connection loopbacks from AC source ports to AC destination ports of the current design. ACLoopbacks specify port connections through a coupling capacitor, as opposed to Loopbacks, which connect directly source ports to destination ports.

Usage Conditions

Note the following:

- *source_port* and *destination_port* must be output and input direction respectively. Inout ports are not allowed.
- 1149.1 ports are not allowed.
- To loop back differential pairs, you must only specify the positive leg's name. The negative leg will be inferred from the BSDL file or taken from the *differential_inverse_of* properties of the ports.

Arguments

- *destination_port : source_port* ;

This port pair describes a connection through a coupling capacitor that is to be made in the simulation test bench and assumed by the test patterns.

Examples

In the example, OUT_AC1 and OUT_AC1_DV are looping back to uncontacted ports IN_AC1 and IN_AC1_DV respectively. Because IN_AC1_DV and OUT_AC1_DV are differential ports, their negative leg will also automatically be looped back. The resulting Verilog coupling capacitors will wait 15 ns before setting the loopback signal going into the inputs to Z.

```
LoadBoardInfo(tester1) {
    [...]
    dot6_ttest : 15.0e-09s ;
    ACLoopbacks {
        IN_AC1 : OUT_AC1 ;
        IN_AC1_DV : OUT_AC1_DV ;
    }
}
```

InSystemTest

Specifies the insertion of an IST controller inside the PatternsSpecification wrapper

Usage

```
PatternsSpecification (design_name, design_id, pattern_id) {
    InSystemTest {
        Controller(icl_instance_name) {
            TestProgram(index) {
                // TestProgram is a repeatable wrapper, where index starts at 0
                pattern: pattern_wrapper_name;
            }
        }
    }
}
```

Description

InSystemTest is an optional wrapper specified within the PatternsSpecification used to generate patterns for an IST controller. By default—that is, usage is set to simulation—the tool generates both a Verilog testbench and memory contents.

If you specify manufacturing_test as the usage, for the direct memory access architecture, the tool generates the memory contents file. For the CPU-based access architecture, it populates the mentor::in_system_test::PatternsSpecification dictionary.

The Controller wrapper is repeatable, depending on how many IST controllers you defined with the DftSpecification/InSystemTest wrapper.

The *icl_instance_name* is an existing ICL instance name. For details about Tessent MissionMode, refer to the [Tessent MissionMode User's Manual](#).

Arguments

- TestProgram (*index*) {
 pattern : *pattern_wrapper_name* ;
}

A repeatable wrapper that specifies a single test program to be executed by the IST controller. For the CPU-based access controller, you can specify any number of test programs. For the direct memory access controller, you can specify a maximum of max_test_program_count wrappers.

The *index* is an integer value starting at 0 that represents the test program number. For direct memory access, you can specify a range for the *index* from 0 to max_test_program_count 1. For CPU's, any number of test programs are allowed.

The pattern:*pattern_wrapper_name* property specifies the existing pattern wrapper name that is applied by the controller. Properties such as test_period, tck_ratio, constant port settings, and clock periods for the test program are sourced from the specified pattern.

Examples

The following example shows a dofile snippet for the In-System Test patterns specification after inserting three IST controllers.

```
...
PatternsSpecification(top, gate, ist) {
    Patterns(m8051_A_ist) {
        # test steps here ...
    }
    Patterns(m8051_B_ist) {
        # test steps here ...
    }
    Patterns(piccpu_ist) {
        # test steps here ...
    }
    InSystemTest {
        Controller(top_gate_tessent_in_system_test_m8051_A_inst) {
            TestProgram(0) {
                pattern: m8051_A_ist;
            }
        }
        Controller(top_gate_tessent_in_system_test_m8051_B_inst) {
            TestProgram(0) {
                pattern: m8051_B_ist;
            }
        }
        Controller(top_gate_tessent_in_system_test_piccpu_inst) {
            TestProgram(0) {
                pattern: piccpu_ist;
            }
        }
    }
}
process_patterns_specification
...
```

DefaultsSpecification Configuration Syntax

This section describes the configuration data syntax used inside Tessent Shell to specify default values used by the `create_dft_specification` command. Tessent Shell comes with built-in default values for all configuration properties. You can use the `DefaultsSpecification` wrapper to change many of those default values to match your preferences and requirements.

For complete information, see the “[DftSpecification](#)” on page 3179

DefaultsSpecification	3686
DefaultsSpecification/DftSpecification	3687
IjtagNetwork	3689
BoundaryScan	3693
MemoryBISR	3699
MemoryBIST	3703
DefaultsSpecification/PatternsSpecification	3714
SignOffOptions	3715
ManufacturingOptions	3717

DefaultsSpecification

Specifies changes to the built-in default values of all configuration properties

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
    DftSpecification {
    }
    PatternsSpecification {
    }
}
```

Description

Specifies changes to the built-in default values of all configuration properties

You can use the DefaultsSpecification wrapper to change many of the built-in default values to match your preferences and requirements. The defaults setting mechanism also comes with a hierarchy of DefaultsSpecification wrappers that allow you to have company, group, and user-level defaults. You can load three configuration files in your .tesson_startup file: one containing a DefaultsSpecification(company) wrapper, one containing a DefaultsSpecification(group) wrapper, and one containing a DefaultsSpecification(user) wrapper. The final default value of a given property is taken from the DefaultsSpecification(user) wrapper if it is specified in that wrapper; it is taken from the DefaultsSpecification(group) wrapper if it is defined in that wrapper but not in the DefaultsSpecification(user) wrapper; and it is taken from the DefaultsSpecification(company) wrapper when it is defined there but not in the DefaultsSpecification(group) and DefaultsSpecification(user) wrappers. The final default value is the built-in default value if it is not specified in any of the three DefaultsSpecification wrappers.

You can refer to the [read_config_data](#) command to learn about loading in configuration data. You can refer to the [get_defaults_value](#) command to learn about accessing the specified defaults values from your dofile. If you prefer, you can also use the “[set_defaults_value -policy](#)” command to define your different policies instead of using configuration files.

Arguments

None

DefaultsSpecification/DftSpecification

A wrapper that specifies options that define the default values of the properties in the DFTSpecification wrapper.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
    DftSpecification {
        rtl_extension           : string ; // default: v
        gate_extension          : string ; // default: vg
        reuse_modules_when_possible : boolean ; // default: auto
        use_rtl_cells           : boolean ; // default: auto
        dft_cell_selection_name : string ;
        persistent_clock_cell_prefix : string ;
            //default: tesson_persistent_cell_
        persistent_cell_prefix   : string ;
            //default: tesson_persistent_cell_
        use_rtl_synchronizer_cell : on | off | auto ; // default: auto
        use_synchronizer_cell_with_reset : on | off ; // default: on
        IjtagNetwork {
        }
        BoundaryScan {
        }
        MemoryBist {
        }
        MemoryBisr {
        }
    }
}
```

Description

A wrapper that specifies options that define the default values of the properties in the DFTSpecification wrapper.

Arguments

- **rtl_extension : string ;**
A string that specifies the default value for the corresponding property found inside the **DftSpecification** wrapper.
- **gate_extension : string ;**
A string that specifies the default value for the corresponding property found inside the **DftSpecification** wrapper.
- **reuse_modules_when_possible : boolean ;**
A string that specifies the default value of the corresponding property found inside the **DftSpecification** wrapper.
- **use_rtl_cells : boolean ; // default: auto**
A property that sets the default value of the use_rtl_cells property found inside the **DftSpecification** wrapper when the **create_dft_specification** command is used.

- `dft_cell_selection_name : string ;`
A property that sets the default value of the `dft_cell_selection_name` property found inside the `DftSpecification` wrapper when the `create_dft_specification` command is used.
- `persistent_clock_cell_prefix : string ; // default: tesson_persistent_cell_`
A property that sets the default value of the `persistent_clock_cell_prefix` property found inside the `DftSpecification` wrapper when the `create_dft_specification` command is used.
- `persistent_cell_prefix : string ; // default: tesson_persistent_cell_`
A property that sets the default value of the `persistent_cell_prefix` property found inside the `DftSpecification` wrapper when the `create_dft_specification` command is used.
- `use_rtl_synchronizer_cell : on | off | auto ; //default: auto`
A property that sets the default value of the `use_rtl_synchronizer` property found inside the `DftSpecification` wrapper when the `create_dft_specification` command is used. This property is currently only used by the OCC and the MemoryBist wrappers.
- `use_synchronizer_cell_with_reset : on | off ; //default: on`
A property that sets the default value of the `use_synchronizer_cell_with_reset` property found inside the `DftSpecification` wrapper when the `create_dft_specification` command is used.

Examples

The following example shows a GlobalOptions wrapper that specifies the `rtl_extension` as “vb” and the `gate_extension` as ”v” inside the group defaults policy. Because it is not overridden by a user defaults policy wrapper, those default values are used by the `create_dft_specification` command.

```
> read_config_data -from_string {
  DefaultsSpecification(user) {
    DftSpecification {
      rtl_extension : vb ;
      gate_extension : v ;
    }
  }
}
> get_defaults_value DftSpecification/rtl_extension
vb
```

IjtagNetwork

A wrapper that is used to specify the default value of properties found in the DftSpecification/IjtagNetwork wrapper and define how the ijtag instances are to be handled during [create_dft_specification](#).

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
    DftSpecification {
        IjtagNetwork {
            connect_data_signals_already_connected_to_ports : on | off ;
            ImplementationOptions {
                scan_path_retiming : latch | flop ;
            }
            HostScanInterface {
                Block {
                    tck           : port_name ; // default: ijtag_tck
                    reset         : port_name ; // default: ijtag_reset
                    select        : port_name ; // default: ijtag_sel
                    shift_en      : port_name ; // default: ijtag_se
                    capture_en   : port_name ; // default: ijtag_ce
                    update_en    : port_name ; // default: ijtag_ue
                    scan_in       : port_name ; // default: ijtag_si
                    scan_out      : port_name ; // default: ijtag_so
                }
                Chip {
                    tck           : port_name ; // default: tck
                    tdi           : port_name ; // default: tdi
                    tdo           : port_name ; // default: tdo
                    tms           : port_name ; // default: tms
                    trst          : port_name ; // default: trst
                }
            }
        }
    }
}
```

Description

A wrapper that is used to specify the default value of properties found in the DftSpecification/IjtagNetwork wrapper and define how the ijtag instances are to be handled during [create_dft_specification](#).

The properties found in the HostScanInterface/Block wrapper are used to define the Interface of the HostScanInterface when the design level is set to “physical_block” or “sub_block” using the [set_design_level](#) command. The properties found in the HostScanInterface/Chip wrapper are used to define the Interface of the HostScanInterface when the design level is set to “chip”.

Arguments

- connect_data_signals_already_connected_to_ports : on | off ;

A property that defines how the [create_dft_specification](#) command should deal with ijtag instances that have their

`set_ijtag_instance_options -connect_data_signals_already_connected_to_ports`
option set to auto.

- `ImplementationOptions/scan_path_retiming : latch | flop ;`
A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.
- `HostScanInterface/Block/tck : port_name ;`
A property that sets the default value of the tck property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during `create_dft_specification` when design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/reset : port_name ;`
A property that sets the default value of the reset property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during `create_dft_specification` when design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/select : port_name ;`
A property that sets the default value of the select property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the `create_dft_specification` command when design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/shift_en : port_name ;`
A property that sets the default value of the shift_en property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the `create_dft_specification` command when the design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/capture_en : port_name ;`
A property that sets the default value of the capture_en property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the `create_dft_specification` command when the design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/update_en : port_name ;`
A property that sets the default value of the update_en property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the `create_dft_specification` command when the design level is `physical_block` or `sub_block`.
- `HostScanInterface/Block/scan_in : port_name ;`
A property that sets the default value of the scan_in property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the `create_dft_specification` command when the design level is `physical_block` or `sub_block`.

- HostScanInterface/Block/scan_out : *port_name* ;
A property that sets the default value of the scan_out property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.
- HostScanInterface/Chip/tck : *port_name* ;
A property that sets the default value of the tck property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is chip.
- HostScanInterface/Chip/trst : *port_name* ;
A property that sets the default value of the trst property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is chip.
- HostScanInterface/Block/tms : *port_name* ;
A property that sets the default value of the tms property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is chip.
- HostScanInterface/Block/tdi : *port_name* ;
A property that sets the default value of the tdi property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is chip.
- HostScanInterface/Block/tdo : *port_name* ;
A property that sets the default value of the tdo property found inside the [HostScanInterface/Interface](#) wrapper of the DftSpecification/[IjtagNetwork](#) wrapper. The default value is used during the create_dft_specification command when the design level is chip.

Examples

The following wrappers define custom naming convention for the IJTAG ports that serves as the default for the entire company:

```
DefaultsSpecification(company) {
    DftSpecification {
        IjtagNetwork {
            HostScanInterface {
                Block {
                    tck           : my_tck ;
                    reset         : my_reset ;
                    select        : my_select ;
                    shift_en      : my_shift_en ;
                    capture_en   : my_capture_en ;
                    update_en    : my_update_en ;
                    scan_in       : my_scan_in ;
                    scan_out      : my_scan_out ;
                }
                Chip {
                    tck           : my_tck ;
                    tdi           : my_tdi ;
                    tdo           : my_tdo ;
                    tms           : my_tms ;
                    trst          : my_trst ;
                }
            }
        }
    }
}
```

BoundaryScan

Specifies certain default values for the boundary scan chain and embedded boundary scan chain to build and optionally insert into the design.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        BoundaryScan {
            outputs_per_enable_cell      : int ; // default: 16
            max_segment_length_for_logictest : int | unlimited | auto ;
            tck_period                   : period ; // default 100ns
            Interface {
            }
            ImplementationOptions {
            }
        }
    }
}
```

Description

A wrapper that specifies certain default values for the both boundary scan and embedded boundary scan. All options apply to both, so is the [ImplementationOptions](#) wrapper. Only the [Interface](#) wrapper applies only to embedded boundary scan.

Arguments

- `outputs_per_enable_cell : int ;`
An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. The default value of Tessent BoundaryScan is 16. Setting a value in the [DefaultsSpecification](#) implies new default values to be used for the respective property of both, the [DftSpecification/BoundaryScan](#) and [DftSpecification/EmbeddedBoundaryScan](#), respectively.
- `max_segment_length_for_logictest : int | unlimited | auto ;`
An optional property that instructs the tool to segment the boundary scan chains into smaller segments to be reused by scan-based logictest modes. The default value of Tessent BoundaryScan is auto. Setting a value in the [DefaultsSpecification](#) implies new default values to be used for the respective property of both, the [DftSpecification/BoundaryScan](#) and [DftSpecification/EmbeddedBoundaryScan](#), respectively.
- `tck_period : period ;`
An optional property that instructs the tool to use a different default period of the test clock. The default value of Tessent BoundaryScan period is 100ns. Setting a value in the [DefaultsSpecification](#) implies new default values to be used for the respective property of both, the [DftSpecification/BoundaryScan](#) and [DftSpecification/EmbeddedBoundaryScan](#), respectively.

Interface

Specifies certain default values for the embedded boundary scan interface only.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
    DftSpecification {
        BoundaryScan {
            Interface {
                select : port_naming ; //bscan_select
                reset : port_naming ; //bscan_reset
                force_disable : port_naming ; //bscan_force_disable
                select_jtag_input : port_naming ; //bscan_select_jtag_input
                select_jtag_output : port_naming ;
                                //bscan_select_jtag_output
                bscan_clock : port_naming ; //bscan_clock
                capture_en : port_naming ; //bscan_capture_en
                shift_en : port_naming ; //bscan_shift_en
                update_en : port_naming ; //bscan_update_en
                scan_in : port_naming ; //bscan_scan_in
                scan_out : port_naming ; //bscan_scan_out
                auxiliary_output : port_naming ; //bscan_%s_aux_out
                auxiliary_output_enable : port_naming ; //bscan_%s_aux_out_en
                auxiliary_input : port_naming ; //bscan_%s_aux_in
                auxiliary_input_enable : port_naming ; //bscan_%s_aux_in_en
                ac_init_clock0 : port_naming ; //bscan_ac_init_clk0
                ac_init_clock1 : port_naming ; //bscan_ac_init_clk1
                ac_signal : port_naming ; //bscan_ac_signal
                ac_mode_en : port_naming ; //bscan_ac_mode_en
            }
        }
    }
}
```

Description

A wrapper that specifies certain default values for the embedded boundary scan interface to be used. Using this wrapper you can override the default naming convention of the Tessent BoundaryScan tool's embedded boundary scan DFT specification DftSpecification/ EmbeddedBoundaryScan/Interface.

Arguments

- **select : *port_naming* ;**

An optional property that defines the name of the select signal. The select signal is used to enable or disable the embedded boundary scan interface of the core. When not specified, the port is named “bscan_select”. Setting this value in the **DefaultsSpecification** implies a new default value to be used for the respective property of DftSpecification/ EmbeddedBoundaryScan/Interface.

- **reset : *port_naming* ;**

An optional property used to provide the name for the port carrying the reset function. The reset function is active low. When not specified, the port is named “bscan_reset”. Setting

this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `force_disable : port_naming ;`

An optional property used to provide the name for the port carrying the `force_disable` function. When not specified, the port is named “bscan_force_disable”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `select_jtag_input : port_naming ;`

An optional property used to provide the name for the port carrying the `select_jtag_input` function. When not specified, the port is named “bscan_select_jtag_input”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `select_jtag_output : port_naming ;`

An optional property used to provide the name for the port carrying the `select_jtag_output` function. When not specified, the port is named “bscan_select_jtag_output”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `bscan_clock : port_naming ;`

An optional property used to provide the name for the port carrying the `bscan_clock` function. When not specified, the port is named “bscan_clock”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `capture_en : port_naming ;`

An optional property used to provide the name for the port carrying the `capture_en` function. When not specified, the port is named “bscan_capture_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `shift_en : port_naming ;`

An optional property used to provide the name for the port carrying the `shift_en` function. When not specified, the port is named “bscan_shift_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `update_en : port_naming ;`

An optional property used to provide the name for the port carrying the `update_en` function. When not specified, the port is named “bscan_update_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- `scan_in : port_naming ;`
An optional property used to provide the name for the port carrying the `scan_in` function. When not specified, the port is named “bscan_scan_in”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.
- `scan_out : port_naming ;`
An optional property used to provide the name for the port carrying the `scan_out` function. When not specified, the port is named “bscan_scan_out”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.
- `auxiliary_output : port_naming ;`
An optional property used to provide the name for the port carrying the `auxiliary_output` function. When not specified, the port is named “bscan_%s_aux_out”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the `port_name` the auxiliary output is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output port associated with port GPIO[4] is “bscan_GPIO_4_aux_out”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.
- `auxiliary_output_enable : port_naming ;`
An optional property used to provide the name for the port carrying the `auxiliary_output_enable` function. When not specified, the port is named “bscan_%s_aux_out_en”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the `port_name` the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output enable port associated with port GPIO[4] is “bscan_GPIO_4_aux_out_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.
- `auxiliary_input : port_naming, ... ;`
An optional property used to provide the name for the port carrying the `auxiliary_input` function. When not specified, the port is named “bscan_%s_aux_in”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the `port_name` the auxiliary input is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input port associated with port GPIO[4] is “bscan_GPIO_4_aux_in”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.
- `auxiliary_input_enable : port_naming, ... ;`
An optional property used to provide the name for the port carrying the `auxiliary_input_enable` function. When not specified, the port is named

“bscan_%s_aux_in_en”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input enable port associated with port GPIO[4] is “bscan_GPIO_4_aux_in_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_init_clock0 : *port_naming* ;

An optional property used to provide the name for the port carrying the ac_init_clock0 function, which has a low “off” state. When not specified, the port is named “bscan_ac_init_clk0”. This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_init_clock1 : *port_naming* ;

An optional property used to provide the name for the port carrying the ac_init_clock1 function, which has a high “off” state. When not specified, the port is named “bscan_ac_init_clk1”. This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_signal : *port_naming* ;

An optional property used to provide the name for the port carrying the ac_signal function. When not specified, the port is named “bscan_ac_signal”. This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_mode_en : *port_naming* ;

An optional property used to provide the name for the port carrying the ac_mode_en function. When not specified, the port is named “bscan_ac_mode_en”. This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification(module_name,id) {
        BoundaryScan {
            ImplementationOptions {
                clocking : tck | gated_tck | gated_tck_inv ; /*DefSpec
                update_stage : flop | latch ; /*DefSpec
                scan_path_retiming : flop | latch ; /*DefSpec
            }
        }
    }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- *clocking* : tck | gated_tck | gated_tck_inv ;
An optional property that specifies how the clocking of the boundary-scan cell should be constructed. The default value of Tessent BoundaryScan is gated_tck. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.
- *update_stage* : flop | latch ;
An optional property that specifies if the update stage is to be implemented using a latch or a flop. The default value of Tessent BoundaryScan is latch. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.
- *scan_path_retiming* : flop | latch ;
An optional property that specifies if the scan path retiming element is to be implemented using a latch. The default value of Tessent BoundaryScan is latch. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.

MemoryBISR

Specifies certain default values for the memoryBISR to build and optionally insert into the design.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBISR {
            insert_fuse_box_controller_in_sub_block : boolean ; // default: off
            insert_fuse_box_controller_in_chip : boolean ; // default: on
            insert_fuse_box_controller_in_physical_block : boolean ;
                                            // default: off
            max_fuse_box_programming_sessions : int | unlimited ;
                                            // default: 1
            fuse_box_location : internal | external ;
            ChainInterface {
                scan_in : port_naming ; // Default : %s_bisr_si
                parallel_in : port_naming ;
                               // Default : %s_bisr_parallel_in
                scan_out : port_naming ; // Default : %s_bisr_so
                serial_repair_enable: port_naming ;
                               // Default : %s_bisr_serial_repair_enable
                capture_shift_clock : port_naming ; // Default : %s_bisr_clk
                shift_en : port_naming ; // Default : %s_bisr_shift_en
                reset : port_naming ; // Default : %s_bisr_reset
                memory_disable : port_naming ;
                               // Default : %s_bisr_mem_disable
                memory_chain_select : port_naming ;
                               // Default : %s_bisr_mem_chain_select
            }
        }
    }
}
```

Description

A wrapper that specifies certain default values for the memoryBISR to build and optionally insert into the design.

Arguments

- insert_fuse_box_controller_in_chip : on | off | when_fuse_box_present ;
 insert_fuse_box_controller_in_sub_block : on | off | when_fuse_box_present ;
 insert_fuse_box_controller_in_physical_block : on | off | when_fuse_box_present ;

Three optional properties that instructs the tool where to place the fuse box controller. The default is ON for the chip level and OFF for both, the sub block and physical block. The value of when_fuse_box_present allows the tool to add a controller at the same level of the fuse box, but only when there is actually a fuse box present at this level. If set, no controller will be added, if there is no fuse box.

There is no equivalent to these properties in the DftSpecification. If you want to change the fuse box controller location, you must use these properties in the DefaultsSpecification.

- `fuse_box_location : internal | external ;`

An optional property that specifies if the fuse box is to be instantiated inside the BISR controller, or if it already exists in the design and connections should be made to and from it. The default value of `fuse_box_location` is `internal`. Setting a value in the `DefaultsSpecification` implies a new default value to be used for the respective property of the `DftSpecification/MemoryBisr/Controller` wrapper.

- `max_fuse_box_programming_sessions : int | unlimited ;`

An optional property that specifies how many fuse box programming sessions are to be supported by the controller. The default value of `max_fuse_box_programming_sessions` is 1. The `unlimited` value disables the compression of repair information. Setting a value in the `DefaultsSpecification` implies a new default value to be used for the respective property of the `DftSpecification/MemoryBisr/Controller` wrapper. Refer to the description of this property in the `DftSpecification` for a more detailed description.

- `ChainInterface` wrapper

An optional wrapper that specifies the names of the ports created when connecting the BISR chains to ports of a module of type `sub_block` or `physical_block` as specified using the `set_design_level` command. A detailed description of the individual ports can be found at the `DftSpecification/MemoryBisr/Interface` wrapper.

- `scan_in : port_naming ;`

A string that specifies the naming to use for the `scan_in` port. When unspecified, it defaults to the value specified in the `MemoryBisr/Interface/scan_in` property inside the `DefaultsSpecification/DftSpecification` wrappers. See the `get_defaults_value` command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `parallel_in : port_naming ;`

A string that specifies the naming to use for the BISR chain parallel input port. This property is only effective when `DftSpecification/MemoryBisr/memory_repair_loading_method: from_read_buffer` has been specified, otherwise it is ignored. When `parallel_in` is unspecified, it defaults to `%s_bisr_parallel_in`.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by the underscore when the module contains more than one power domain name; otherwise it is replaced by an empty string.

- `scan_out : port_naming ;`

A string that specifies the naming to use for the `scan_out` port. When unspecified, it defaults to the value specified in the `MemoryBisr/Interface/scan_out` property inside the `DefaultsSpecification/DftSpecification` wrappers. See the `get_defaults_value` command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `serial_repair_enable : port_naming ;`

A string that specifies the naming to use for the fast BISR chain loading enable port. This property is only effective when DftSpecification/[MemoryBisr](#)/memory_repair_loading_method: from_read_buffer has been specified, otherwise it is ignored. This port will be connected to the “SerialRepairEnable” output port of the fuse box controller. When unspecified, it defaults to “%s_bisr_serial_repair_enable”.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by the underscore when the module contains more than one power domain name; otherwise it is replaced by an empty string.

- `capture_shift_clock : port_naming ;`

A string that specifies the naming to use for the capture_shift_clock port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/Interface/capture_shift_clock property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group, and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `shift_en : port_naming ;`

A string that specifies the naming to use for the shift_en port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/Interface/shift_en property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `reset : port_naming ;`

A string that specifies the naming to use for the active low reset port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/Interface/reset property inside the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name; otherwise, it is replaced by an empty string.

- `memory_disable : port_naming ;`

A string that specifies the naming to use for the memory_disable port. When unspecified, it defaults to the value specified in the [MemoryBisr](#)/Interface/memory_disable property inside

the [DefaultsSpecification/DftSpecification](#) wrappers. See the [get_defaults_value](#) command description for more details about the company, group and user defaults specifications.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by an underscore when the module contains more than one power domain name otherwise, it is replaced by an empty string.

This port is only created when at least one memory instance has a Memory wrapper (see “[Tessent Core Description](#)” on page 3755) with function “bisrSerialData” and a Memory with function “select” in its Memory library description or a child block with this port is present. Such ports are identified in the ICL module as [DataInPorts](#) with the Attribute “tessent_bisr_function” set to “memory_disable”.

- **memory_chain_select : *port_naming* ;**

A string that specifies the naming to use for the BISR register chain select MSEL port. This port will be connected to the “bisrSeIM” output port of the fuse box controller. When unspecified, it defaults to “%s_bisr_mem_chain_select”.

The string must contain the “%s_” symbol. It is replaced by the power domain name followed by the underscore when the module contains more than one power domain name; otherwise it is replaced by an empty string.

MemoryBist

Specifies the default values of properties found in the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            clock_partitioning      : per_clock_domain | per_sync_clock_group ;
            max_steps_per_controller : int | unlimited ;
            max_test_time_per_controller : time | 500ms ; // Default is 500ms
            max_memories_per_step    : int | unlimited ;
            max_power_per_step       : int | unlimited ; // Default is 500mW
            single_memory_dimension_per_step : on | off ;
            ControllerOptions {
            }
            RepairOptions {
            }
            DiagnosisOptions {
            }
            AlgorithmResourceOptions {
            }
            MemoryInterfaceOptions {
            }
            MemoryClusterOptions {
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/
[MemoryBist](#) wrapper.

You can also provide test scheduling constraints to guide the creation of the DftSpecification using the [create_dft_specification](#) command.

Arguments

- clock_partitioning : per_clock_domain | per_sync_clock_group ;

A property that specifies how memories will be grouped by the [create_dft_specification](#) command according to their clock sources. When set to per_clock_domain, the memories on the same clock domain are allowed to be assigned to a controller. When set to per_sync_clock_group, controller sharing is allowed for memories belonging to synchronous clock sources and having the same clock frequency. Synchronous clock groupings are specified with the [add_synchronous_clock_group](#) command and current groupings can be reported with the [report_synchronous_clock_groups](#) command.

- max_steps_per_controller : int | unlimited ;

A property that constrains the number of test steps that will be allocated per controller. A controller step designates the group of memories that will be tested in parallel. The

[create_dft_specification](#) command will assign test steps to controllers such that the requirement is met.

When running the [process_dft_specification](#) command, the validation of the DftSpecification will issue a warning if the controller step count requirement is exceeded.

- max_test_time_per_controller : time | unlimited ;

A property that constrains the test time that a controller can use to complete the test. The [create_dft_specification](#) command will assign memories to controllers such that the requirement is met. The default value is 500ms.

The estimated test time of a controller is the sum of the execution time of its controller steps. The test time of a controller step depends on factors such as the chosen algorithm and the memory sizes.

When running the [process_dft_specification](#) command, the validation of the DftSpecification will issue a warning if the test time requirement is exceeded.

- max_memories_per_step : int | unlimited ;

A property that constrains the number of memories that will be tested in parallel within a controller step. The [create_dft_specification](#) command will assign memories to controller steps such that the requirement is met.

When running the [process_dft_specification](#) command, the validation of the DftSpecification will issue a warning if the memory count requirement is exceeded.

- max_power_per_step : int | unlimited ;

A property that constrains the power consumption per controller step. The [create_dft_specification](#) command will assign memories to controller steps such that the requirement is met. The specified value is assumed to be in milliwatts. The default value is 500mW.

The estimated power per step is the sum of the power consumed by memories tested in parallel. The memory power is calculated by multiplying the MilliWattsPerMegaHertz property in the memory library description by the frequency of the clock domain. It does not include the power consumed by the BIST circuitry.

When running the [process_dft_specification](#) command, the validation of the DftSpecification will issue a warning if the power consumption requirement is exceeded.

- single_memory_dimension_per_step : on | off ;

A property that specifies how memories of different dimensions will be grouped by the [create_dft_specification](#) command.

When set to on, only memories having the same number of columns, rows, and banks will be allowed to be tested in parallel within a controller step. When set to off, memories of different sizes will be allowed to be tested in parallel.

ControllerOptions

Specifies the default values of properties found in the Controller/AdvancedOptions and Controller/Step wrappers of the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            ControllerOptions {
                algorithm : algo_name ; // Default: // from library
                operation_set : opset_name ;// Default: // from library
                extra_algorithms : algo_name, ... ;
                extra_operation_sets : opset_name, ... ;
                functional_debug_mode : on | off ;
                comparator_location : shared_in_controller | per interface ;
                pipeline_controller_outputs : on | off ;
                selective_parallel_memory_test : on | off ;
                shared_comparators_per_go_id : int | all ; // Default: 1
                use_multicycle_paths : on | off ;
                gate_bist_clock_in_functional_mode : off | with_clock_gating_cell | with_clock_mux ;
                min_misr_segment_bits : 24 | 256 ;
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the [AdvancedOptions](#) and [Step](#) wrappers of the DftSpecification/MemoryBist wrapper.

Arguments

- `algorithm : algo_name ;`
A property that sets the default value of the algorithm property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- `operation_set : opset_name ;`
A property that sets the default value of the operation_set property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- `extra_algorithms : algo_name, ... ;`
A property that sets the default value of the extra_algorithms property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- extra_operation_sets : *opset_name*, ... ;
A property that sets the default value of the extra_operation_sets property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- functional_debug_mode : on | off ;
A property that sets the default value of the functional_debug_mode property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper
- comparator_location : shared_in_controller | per_interface ;
A property that sets the default value of the comparator_location property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- pipeline_controller_outputs : on | off ;
A property that sets the default value of the pipeline_controller_outputs property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- selective_parallel_memory_test : on | off ;
A property that sets the default value of the selective_parallel_memory_test property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- shared_comparators_per_go_id : *int* | all ;
A property that sets the default value of the shared_comparators_per_go_id property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper. The default is 1.
- use_multicycle_paths : on | off ;
A property that sets the default value of the use_multicycle_paths property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- gate_bist_clock_in_functional_mode : off | with_clock_gating_cell | with_clock_mux ;
A property that sets the default value of the gate_bist_clock_in_functional_mode property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- min_misr_segment_bits : 24 | 256 ;
A property that sets the default value of the min_misr_segment_bits property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

RepairOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/
RepairOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            RepairOptions {
                fuse_set_extraction_sequence : address_before_fuse_map | fuse_map_before_address ;
                row_bira_location : controller | follow_comparators ;
                max_repair_group_size : unlimited | {int[kilobits | megabits]} ;
                repair_sharing : on | off ;
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/
MemoryBist/[RepairOptions](#) wrapper.

Arguments

- **fuse_set_extraction_sequence : address_before_fuse_map | fuse_map_before_address ;**
A property that sets the default value of the fuse_set_extraction_sequence property found inside the [RepairOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **row_bira_location : controller | follow_comparators ;**
A property that sets the default value of the row_bira_location property found inside the [RepairOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **max_repair_group_size : unlimited | {int[kilobits | megabits]}**
A property that specifies the total memory size limit on a memory group. The default value of “unlimited” indicates that any number of compatible memories can be assigned to repair groups. The int[units] pair specifies the maximum size allowed for a repair group. Any memory that would cause this limit to be exceeded will be assigned to a different repair group.
- **repair_sharing on | off**
A property that will globally enable and disable repair sharing for memories that have “-repair_sharing auto” specified, which is the default. The “-repair_sharing” value can be changed from “auto” for a memory instance with the [set_memory_instance_options](#) command.

DiagnosisOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/
DiagnosisOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            DiagnosisOptions {
                comparator_selection_mux      : on | off | auto ;
                go_status                      : auto | per_memory ;
                StopOnErrorOptions {
                    failure_limit             : int | off ; // Default: 4096
                }
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/
MemoryBist/[DiagnosisOptions](#) wrapper.

Arguments

- **comparator_selection_mux** : on | off | auto;
A property that sets the default value of the comparator_selection_mux property found inside the [DiagnosisOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **go_status** : auto | per_memory ;
A property that sets the default value of the go_status property found inside the [DiagnosisOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **StopOnErrorOptions/failure_limit** : *int* | off ;
A property that sets the default value of the StopOnErrorOptions/failure_limit property found inside the [DiagnosisOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper. The default value becomes 4096 when the controller tests at least one memory of type SRAM or DRAM. It becomes “off” when the controller tests only memories of type ROM. The error counter is no longer used for ROM diagnosis as it was in the LV flow. Instead the functional debug mode is used to perform ROM diagnosis.

AlgorithmResourceOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/[AlgorithmResourceOptions](#) wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            AlgorithmResourceOptions {
                soft_instruction_count : int //Default: 0
                data_register_bits : int | auto ;
                counter_a_bits : int | auto ;
                delay_counter_bits : int | auto ;
                max_data_inversion_address_bit_index : int | max_index ;
                // Default: 0
                a_equals_b_command_allowed : on | off ;
                address_segment_x0_y0_allowed : on | off ;
                max_x0_segment_bits : 1 | auto ;
                max_y0_segment_bits : 1 | auto ;
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist/[AlgorithmResourceOptions](#) wrapper.

Arguments

- `soft_instruction_count : int ;`
A property that sets the default value of the `soft_instruction_count` property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/MemoryBist wrapper.
- `data_register_bits : int | auto ;`
A property that sets the default value of the `data_register_bits` property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/MemoryBist wrapper.
- `counter_a_bits : int | auto ;`
A property that sets the default value of the `counter_a_bits` property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/MemoryBist wrapper.
- `delay_counter_bits : int | auto ;`
A property that sets the default value of the `delay_counter_bits` property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/MemoryBist wrapper.

- max_data_inversion_address_bit_index : int | max_index ;
A property that sets the default value of the max_data_inversion_address_bit_index property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper. The default is 0.
- a_equals_b_command_allowed : on | off ;
A property that sets the default value of the a_equals_b_command_allowed property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- address_segment_x0_y0_allowed : on | off ;
A property that sets the default value of the address_segment_x0_y0_allowed property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- max_x0_segment_bits : 1 | auto ;
A property that sets the default value of the max_x0_segment_bits property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- max_y0_segment_bits : 1 | auto ;
A property that sets the default value of the max_y0_segment_bits property found inside the [AlgorithmResourceOptions](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

MemoryInterfaceOptions

Specifies the default values of properties found in the Controller/AdvancedOptions Controller/Step, and Controller/Step/MemoryInterface wrappers of the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            MemoryInterfaceOptions {
                bist_data_in_pipelining      : on | off | int ; // Default: off
                bist_data_out_pipelining     : on | off | per_port ;
                                            // Default: off
                local_comparators_per_go_id  : int | all ; // Default: 1
                observation_xor_size         : 1..8 | off ; // Default: 3
                repair_analysis_present      : off | auto ;
                scan_bypass_logic            : async_mux | none | sync_mux |
                                                from_library ;
                repair_group_scope           : physical_memory | controller ;
                                            // Default: controller
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the [AdvancedOptions](#), [Step](#) and Controller/Step/MemoryInterface wrappers of the DftSpecification/MemoryBist wrapper.

Arguments

- `bist_data_in_pipelining` : `on` | `off` | `int`;

A property that sets the default value of the of the `bist_data_in_pipelining` property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- `bist_data_out_pipelining` : `on` | `off` | `per_port`;

A property that sets the default value of the of the `bist_data_out_pipelining` property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- `local_comparators_per_go_id` : `int` | `all`;

A property that sets the default value of the `MemoryInterface/local_comparators_per_go_id` property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- `observation_xor_size` : `1..8` | `off`;

A property that sets the default value of the `observation_xor_size` property found inside the [AdvancedOptions](#) wrapper of the DftSpecification/MemoryBist/[Controller](#) wrapper.

- repair_analysis_present : off | auto;

A property that sets the default value of the MemoryInterface/repair_analysis_present property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- scan_bypass_logic : async_mux | none | sync_mux | [from_library](#) ;

A property that sets the default value of the MemoryInterface/scan_bypass_logic property found inside the [Step](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

- repair_group_scope : physical_memory | [controller](#) ;

Memory instances that have repair sharing enabled, the selected value impacts repair grouping as follows:

- physical_memory — Specifies the repair group is restricted within a physical memory. Memories with multiple [RowSegment](#) and/or [ColumnSegment](#) wrappers will be considered as having a single RowSegment and/or ColumnSegment wrapper for redundancy analysis as long as the memory size does not exceed the size specified by the max_repair_group_size property in the [RepairOptions](#) wrapper.
- controller — Specifies the repair group is restricted to all memories tested by the controller. The repair group cannot contain memories tested by other controllers. Memories are automatically assigned to repair groups as long as the memory size does not exceed the size specified by the max_repair_group_size property in the [RepairOptions](#) wrapper.

Repair sharing can be enabled for memory instances with the [set_memory_instance_options](#) command if the DefaultsSpecification for repair_sharing is “off”.

MemoryClusterOptions

A wrapper that is used to specify default values of properties found in the DftSpecification/MemoryBist/Controller/MemoryCluster wrapper

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    DftSpecification {
        MemoryBist {
            MemoryClusterOptions {
                pipeline_cluster_inputs      : on | off ;
                pipeline_cluster_outputs     : on | off ;
                memory_access_level         : physical | logical | auto ;
                repair_analysis_present     : auto | off ;
                repair_group_scope          : physical_memory | logical_memory | controller ;
                                            // Default: controller
            }
        }
    }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist/[MemoryCluster](#) wrapper.

Arguments

- **pipeline_cluster_inputs** : on | off ;
A property that sets the default value of the `pipeline_cluster_inputs` property found inside the [MemoryCluster](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **pipeline_cluster_outputs** : on | off ;
A property that sets the default value of the `pipeline_cluster_outputs` property found inside the [MemoryCluster](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **memory_access_level** : physical | logical | auto ;
A property that sets the default value of the `memory_access_level` property found inside the [MemoryCluster](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **repair_analysis_present** : auto | off ;
A property that sets the default value of the `repair_analysis_present` property found inside the [MemoryCluster](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.
- **repair_group_scope** : physical_memory | logical_memory | controller ;
A property that sets the default value of the `repair_group_scope` property found inside the [MemoryCluster](#) wrapper of the DftSpecification/[MemoryBist](#) wrapper.

DefaultsSpecification/PatternsSpecification

Specifies certain default values for the pattern generation process through the PatternsSpecification.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    PatternsSpecification {
        tester_period : 100ns ;
        AdvancedOptions {
            parameter_file   : file_path ;
            Parameters {
                parameter_name : parameter_value ; // repeatable
            }
        }
        SignOffOptions {
        }
        ManufacturingOptions {
        }
    }
}
```

Description

These wrappers are used to define how the patterns are to be created when running the `create_patterns_specification` command to create different pattern types.

Arguments

- `tester_period : period ;`
An optional property that defines the period of the tester clock. The default is 100ns. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of PatternsSpecification/Patterns.
- `AdvancedOptions/parameter_file : file_path ;`
An optional that specifies the name of a file containing parameters used to configure the `write_patterns` command. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of PatternsSpecification.
- `AdvancedOptions/Parameters/parameter_name : parameter_value ;`
An optional that specifies parameter-value pairs used to configure the `write_patterns` command. A parameter specified in this wrapper overrides the same parameter specified in the specified `parameter_file`. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of PatternsSpecification

SignOffOptions

Defines how simulation signoff pattern wrappers are to be created when running the `create_patterns_specification` command.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
    PatternsSpecification {
        SignOffOptions {
            simulate_instruments_in_lower_physical_instances : on | off ;
            testbench_module_name : format_string ;
        }
    }
}
```

Description

This wrapper is used to define how the simulation signoff patterns wrapper are to be created when running the `create_patterns_specification` command to create the signoff simulation patterns.

Arguments

- `simulate_instruments_in_lower_physical_instances : on | off ;`

A property that specifies to re-execute the lower physical instance simulations at higher levels when you are using a hierarchical flow. This property has no effect if your current design does not include any lower physical instances.

This property only affects the pattern specification creation when `create_patterns_specification` is called with usage “signoff”. It has no effect for usage “manufacturing”. When specified to “on”, the pattern specification will include `Patterns` wrapper to simulate all tests situated in the top-level module and below the child physical blocks. The `Patterns` wrapper will keep the simulation of the test in each physical block instance separated from the other such that the interface model of the other physical block instances can be used when simulating a given physical block instance in order to manage the simulation performance. If you have declared some modules as “sub_block” modules using the `set_design_level` command, their tests will always be simulated as part of their parent physical module, whether it is the top module or a child physical block. When the property is set to off, only the tests situated in the top module and its child sub_block modules are simulated. The tests situated below the child physical block are omitted.

- `testbench_module_name : format_string ;`

A property that specifies the top-level module name of the test bench in the `patternName.v` file. You can use this property to give a different test bench module name for each test bench so that more than one test bench can be compiled into a common work library for a Verilog simulator.

The default is TB. You can specify %d to mean the module name of the current design or %p to mean the pattern name. The advantage of using TB for all test benches is that you can use the same waveform file to analyze failing simulations in any test bench.

Related Topics

[DefaultsSpecification/PatternsSpecification](#)

ManufacturingOptions

Defines how simulation manufacturing pattern wrappers are to be created when running the create_patterns_specification command.

Usage

```
DefaultsSpecification(policy) { // legal: company group user
    PatternsSpecification {
        ManufacturingOptions {
            max_async_clock_sources      : integer | unlimited ;
            manufacturing_patterns_format : format ;
            compress_pattern_files       : on | off ;
            MemoryBist {
                max_controllers_per_test_step : integer | unlimited ;
            }
        }
    }
}
```

Description

The ManufacturingOptions wrapper is used in the Defaults/PatternsSpecification wrapper.

Arguments

- `max_async_clock_sources : integer | unlimited ;`

A property that specifies how many asynchronous clocks can be active within one Patterns wrapper. Modern testers can generate multiple asynchronous clocks. Use this property to control how many instruments, belonging to different asynchronous frequency groups, can be executed in parallel.

If you specify an integer, Tessent Shell creates memory BIST patterns that use no more than the specified number of asynchronous clocks. Valid integer values are 0 or greater. When you specify a value of 0, all clocks used in the pattern are synchronous. Use the value of 0 when the tester does not have any asynchronous clock generators.

If you specify unlimited, Tessent Shell groups all memory BIST controllers inside of a single pattern. Use this value when the tester is capable of driving an unlimited number of asynchronous clock channels.

- `manufacturing_patterns_format : format ;`

A property that specifies a format for the saved test patterns. Valid formats are wgl, stil, ctl, stil2005, titdl, fjtld, mitdl, or tstl. The default format is stil.

For a description of each format type, see the write_patterns command's format_switch in the *Tessent Shell Reference Manual*.

- `compress_pattern_files : on | off ;`

A property that enables pattern compression.

- MemoryBist/max.controllers_per_test_step : *integer* | unlimited ;

A property that partitions the memoryBIST controllers into many test steps. When an integer is specified, the memoryBIST controller will insert no more than the specified number memoryBIST controllers per test step during create_patterns_specification. This property only affects the memoryBIST pre-repair (if present) and Go/NoGo patterns.

When generating the ParallelRetentionTest patterns, all memoryBIST controllers are inserted inside the same test step, but the number of controllers with the same parallel_retention_group will not exceed the specified value.

If BIRA is present, the CheckRepairNeeded patterns are not affected by this property.

Related Topics

[DefaultsSpecification/PatternsSpecification](#)

Metadata Configuration Syntax

The metadata syntax is used to define all of the available syntax documented in the DftSpecification and DefaultsSpecification/DftSpecification wrappers.

This section describes the configuration metadata syntax used inside Tesson Shell to define all valid configuration syntax. The metadata syntax is also used to create the configuration tree widgets in DFTVisualizer that are used to edit and view the configuration data in the Configuration Data window. For information about loading configuration data in the DFTVisualizer Configuration Data window, see the [add_config_tab](#) and [display_specification](#) commands. Finally, the metadata is used by the [report_config_syntax](#) command to report the available syntax in a human-readable format that is consistent with how the syntax is documented in this reference guide.

You typically do not need to define metadata because all configuration data is fully defined in the tool. You can, however, define your own metadata to define any configuration syntax needed to automate portions of your flows. Both DFTVisualizer and the [report_config_syntax](#) command, which are driven by the metadata, will completely support your syntax.

You can also augment the metadata of the [DftSpecification](#) for your custom needs. The added elements will be ignored by the [process_dft_specification](#) command. You can then add your own code to access the added properties and use it for your custom needs. For example, you may want to package a fully-specified DftSpecification file for an RTL core such that the user of the core can simply add the DFT hardware by running [process_dft_specification](#) with your packaged DftSpecification file. In this file, you may want to define extra information such as clock sources with their periods. After you have done this, you can use configuration data commands, such as the [get_config_value](#) command, to access the information and issue the [add_clocks](#) commands in your flow. You can also add extra information that is accessed by your [process_dft_specification.post_insertion Procedure](#) and have it automatically perform custom actions based on the captured information in the [DftSpecification](#) file.

If you create custom metadata and want it to be loaded into the tool, you must store the metadata file in the <plugin>/metadata directory, where <plugin> is a directory pointed to by the TESSENT_PLUGIN_PATH environment variable, and you must name the metadata file with the extension “.tesson_meta”. If you name the <plugin> directory “tesson_plugin” and place it one level above the bin directory that contains the Tesson Shell executable “tesson”, the tool automatically loads the metadata at invocation unless the TESSENT_PLUGIN_IGNORE_DEFAULT_PATH environment variable is defined.

Wrapper	3721
Property	3730
RepeatableProperty	3735
DataWrapper	3738
CsvWrapper	3741
Id	3743

Value	3747
RowIdElement	3751

Wrapper

A wrapper used in metadata to define the syntax of a wrapper that is usable in configuration data.

Usage

```
Partitions : partition_list ; // Only on first line of file
Wrapper(wrapper_name) {
    Repeatable      : On | Off | Single ;
    IgnoreUnknowns  : On | Off ;
    Usage           : normal | hidden | Env(EnvVar)
                      | AccessFeature(feature_name) ;
    ReferenceableOnly : On | Off ;
    InsertFirst;
    InsertLast; // default behavior
    InsertBefore     : meta_leaf_name ;
    InsertAfter      : meta_leaf_name ;
    MergeableContent : On | Off ;
    Options(Visualizer) {
        cdv_tab_prefix : string ;
        DisplayString   : string ;
    }
    //Everything below is repeatable and optional
    Id(name_id) {
    }
    Property(property_name) {
    }
    RepeatableProperty(property_name) {
    }
    DataWrapper(wrapper_name) {
    }
    CsvWrapper(wrapper_name) {
    }
    Reference = relative_or_absolute_meta_path ;
    ReferenceWithInheritedDefault = relative_or_absolute_meta_path ;
    ReferenceContents = relative_or_absolute_meta_path ;
}
```

Description

A wrapper used in metadata to define the syntax of a wrapper that is usable in configuration data.

The presence of the Partitions property is used to define the partition in which the wrapper, and all of the elements defined within it, are to exist.

Arguments

- Partitions : *partition_list* ;

An optional property that can only be specified on the first line of a file, and that controls in which partition the metadata is to be used when defining syntax. When unspecified, the wrappers and what is defined within them, are defined in the default partition.

The partition name can be specified as “default”, “tcd”, or “default, tcd”. The “default” partition is the one in which the [DftSpecification](#) and [DefaultsSpecification/DftSpecification](#) wrappers exist. The tcd partition is the one in which the files loaded with the [read_core_descriptions](#) command are loaded into. The tcd partition is read-only and configuration data found in it cannot be modified using configuration-editing commands such as the [add_config_element](#) command.

All existing syntax defined in the “tcd” partition within the tool is also defined in the default partition. If you absolutely need to edit a tcd file, you need to load it into the default partition using the [read_config_data](#) command, perform your edits using the editing commands, save it back to the file using [write_config_data](#), and then load it back into the tcd partition using the [read_core_descriptions](#) command. Even if it exists in the default partition, the content of this configuration data is not used by the tool because the tool only looks for tcd data in the tcd partition.

- *wrapper_name*

A required string that defines the name of the wrapper to create. The string must start with a letter and only include letters, numbers, and underscores. The *wrapper_name* must be unique across all *wrapper_names* for root-level wrappers inside a partition, and across all *wrapper_names* and *property_names* found within a common parent wrapper.

When you want to define extra syntax inside an already defined wrapper, you create a wrapper that contains your new elements and has a “/” as the first character. For example, if you wanted to add a new wrapper called MyCustomData inside DftSpecification/IjtagNetwork, you would do it this way:

```
Wrapper(/DftSpecification/IjtagNetwork) {
    Wrapper(MycustomData) {
        ...
    }
}
```

- Repeatable : On | Off | Single ;

A property used to define whether the wrapper is repeatable. Both the Off and Single options mean that the wrapper is not repeatable. The difference between Off and Single is that a wrapper identified as “Repeatable : Single” always exists whether or not it is specified in the configuration data. You can get the value of properties inside a Single wrapper without having to first check if the wrapper exists or not. A “Repeatable : Off” wrapper does not exist until it is specified.

- IgnoreUnknowns : On | Off ;

A property that specifies that wrappers and properties not declared in the metadata of this wrapper are to be tolerated in this wrapper. You should never set this On in wrappers that are not purely machine generated and consumed by a machine as any typos will go unnoticed and will be silently ignored. These unknown elements will not be viewable and editable in DftVisualizer and [report_config_syntax](#) will not report their syntax. It is much safer to define all allowed syntax within a wrapper even if you choose to keep some hidden using the “Usage : hidden;” property which documented next.

- Usage : normal | hidden | Env(EnvVar) | AccessFeature(*feature_name*) ;

A property that controls the visibility of a wrapper. When set to normal, the wrapper is reported by [report_config_syntax](#) and visible in DFTVisualizer. When set to hidden, the wrapper is not reported by [report_config_syntax](#) and is invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands such as [get_config_value](#) if the name of the hidden element is known to the user. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden. When the Usage value is AccessFeature(*feature_name*), the usage value is “normal”, if the *access_code* for the *feature_name* is enabled, otherwise the Usage value is “hidden”.

- ReferenceableOnly : On | Off ;

A property that controls the existence of the wrapper. When set to On, this wrapper definition does not exist when it is defined. It and its children only exist when referenced using the [Reference](#) property . This option is useful to create a wrapper that defines a library of elements to be referenced from other wrappers.

- *InsertFirst* ;

An implicit Boolean property that specifies that the wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which wrappers with “Repeatable : Single” gets displayed in the configuration tabs inside DFTVisualizer.

- *InsertLast* ;

An implicit Boolean property that specifies that the wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which wrappers with “Repeatable : Single” gets displayed in the configuration tabs inside DFTVisualizer.

- InsertBefore : *meta_leaf_name* ;

A property that specifies that the wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a wrapper before Wrapper(IjtagNetwork), you would specify “InsertBefore : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which wrappers with “Repeatable : Single” are displayed in the configuration tabs inside DftVisualizer.

- InsertAfter : *meta_leaf_name* ;

A property that specifies that the wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a

wrapper after Wrapper(IjtagNetwork), you would specify “InsertAfter : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which wrappers with “Repeatable : Single” are displayed in the configuration tabs inside DftVisualizer.

- MergeableContent : On | Off ;
A property that specifies that the content of the wrapper is merged with existing content. When this property is set to on, the content of the wrapper is merged with the existing content when reading in a new copy of the same parent wrapper. For example, if WprA was defined with MergeableContent set to On and a WprA already exist in memory with SubWrapper1. If you read a second copy of WprA which contains SubWrapper2, the content of the new WprA wrapper will be merged with the content of the one that already exists in memory such that both SubWrapper1 and SubWrapper2 will exist in WprA.
- Options(Visualizer)/cdv_tab_prefix : *string* ;
A property used to control the name of the Tab that is used by DFTVisualizer when the [add_config_tab](#) command is issued on this wrapper. The specified prefix is concatenated with an integer incremented from 1 for all added configuration tabs having a common prefix.
- Options(Visualizer)/DisplayString : *string* ;
A property used to control how the wrapper is labelled in the Configuration Data window. When unspecified, it default to wrapper_name. This option is useful when you want to create a GUI to display configuration data and you want to name the tree nodes using sentences as oppose to the strict wrapper names.
- Reference = *relative_or_absolute_meta_path* ;
A property used to define an element inside the wrapper using the same definition that was used inside another wrapper. This not only allows you to keep the metadata DRY, it also enables the element to be moved or copied from one location to the other using DFTVisualizer or the “[add_config_element](#) -copy_from” and the [move_config_element](#) commands. For an element to be copied or moved from one wrapper to the next, the metadata definition in both places must be created in one place by either having both of them reference a unique definition, or by having one definition reference the other. The path may be absolute or relative such as in these two examples:

```
Reference = ../Interface/scan_in;
Reference = /DftSpecification/IjtagNetwork/HostScanInterface/
Interface/scan_in;
```

- ReferenceWithInheritedDefault = *relative_meta_path* ;
A property used to define a property inside the wrapper using the same definition that was used inside another wrapper. Not only is the definition inherited, the value of the referenced property becomes the default value of the new property. To use this feature, the *relative_meta_path* can only come back down in wrappers specified with “Repeatable :

Single”; this is because to be able to inherit the default value from it, the referenced property must always exist whenever the referencing property exists. For example, the path `../../Interface/scan_in` is legal because Interface is defined with “Repeatable : Single” inside the IJtagNetwork/HostScanInterface wrapper. The above example could be used to create a property called “scan_in” in a wrapper found inside HostScanInterface that defaults to the value of the “`../../Interface/scan_in`” property when the “`./scan_in`” property is not specified.

- ReferenceContents = *relative_or_absolute_meta_path* ;

A property used to reference the elements inside of the wrapper that it points to and not the wrapper itself. See [Example 2](#) for an example usage of this property.

Examples

Example 1

The following example defines the metadata used by the Sib wrapper inside the DftSpecification/IJtagNetwork wrapper. Following is the constructed syntax reported by report_config_syntax on the same wrapper. Notice the recursive definition of the Sib node inside the Sib node achieved by referencing itself using “Reference = `../Sib;`”.

```
> report_config_data -partition meta \
    /dftspecification/ijtagnetwork/hostscaninterface/sib
```

```
Wrapper(Sib) {
    Repeatable : On;
    Id(id) {
        Type : String;
        Repeatable : Off;
        IgnoreCase : On;
        Options(Visualizer) {
            Counter : global;
            DefaultValueList : #;
        }
    }
    Wrapper(Interface) {
        Repeatable : single;
        Wrapper(Client) {
            Repeatable : single;
            Property(tck) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_tck;
                }
            }
            Property(reset) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_reset;
                }
            }
            Property(select) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_sel;
                }
            }
            Property(capture_en) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_ce;
                }
            }
            Property(shift_en) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_se;
                }
            }
            Property(update_en) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_ue;
                }
            }
            Property(scan_in) {
                Value(port_name) {
                    Type : String;
                    DefaultValue : ijtag_si;
                }
            }
            Property(scan_out) {
```

```
        Value(port_name) {
            Type : String;
            DefaultValue : ijtag_so;
        }
    }
    Property(reset_polarity) {
        Options(Visualizer) {
            BlankSpaceAbove : on;
        }
        Value(active_polarity) {
            Type : String;
            LegalString : active_high, active_low;
            DefaultValue : active_low;
            IgnoreCase : On;
        }
    }
}
Wrapper(Host) {
    Repeatable : single;
    Property(select) {
        Value(port_name) {
            Type : String;
            DefaultValue : ijtag_to_sel;
        }
    }
    Property(scan_in) {
        Value(port_name) {
            Type : String;
            DefaultValue : ijtag_from_so;
        }
    }
}
Reference = ../ScanMux/Attributes;
Reference = ../ScanMux/parent_instance;
Reference = ../ScanMux/leaf_instance_name;
Reference = ../ScanMux/Input/keep_active_during_scan_test;
Property(capture_value) {
    Value(enum) {
        type : string;
        LegalString : 1, 0, self;
        DefaultValue : 0;
    }
}
Reference = ../Sib;
Reference = ../Tdr;
Reference = ../ScanMux;
Reference = ../DesignInstance;
}

> report_config_syntax \
    /dftspecification/ijtagnetwork/hostscaninterface/sib
```

```
DftSpecification(<module_name>,<id>) {
    IJtagNetwork {
        HostScanInterface(<id>) {
            Sib(<id>) {
                Interface {
                    Client {
                        tck : <port_name> ; // default: ijtag_tck
                        reset : <port_name> ; // default: ijtag_reset
                        select : <port_name> ; // default: ijtag_sel
                        capture_en : <port_name> ; // default: ijtag_ce
                        shift_en : <port_name> ; // default: ijtag_se
                        update_en : <port_name> ; // default: ijtag_ue
                        scan_in : <port_name> ; // default: ijtag_si
                        scan_out : <port_name> ; // default: ijtag_so

                        reset_polarity : <active_polarity> ; // legal : active_high
(active_low)
                    }
                    Host {
                        select : <port_name> ; // default: ijtag_to_sel
                        scan_in : <port_name> ; // default: ijtag_from_so
                    }
                }
                Attributes {
                    <attribute> : <string> ; // repeatable
                }
                parent_instance : <instance_name> ;
                leaf_instance_name : <leaf_instance_name> ;
                keep_active_during_scan_test : <boolean> ; // default: auto
                capture_value : <enum> ; // legal : 1 (0) self
            }
            Sib(<id>) {
                // Same as /DftSpecification/IJtagNetwork/HostScanInterface/Sib
            }
            Tdr(<id>) {
                // Same as /DftSpecification/IJtagNetwork/HostScanInterface/Tdr
            }
            ScanMux(<id>) {
                // Same as /DftSpecification/IJtagNetwork/HostScanInterface/
            }
            ScanMux
        }
        DesignInstance(<instance_name>) {
            // Same as /DftSpecification/IJtagNetwork/HostScanInterface/
        }
        DesignInstance
    }
}
}
```

Example 2

In the following example, the two “Test” wrappers are equivalent. In the first case, the objects P and Q of the RefWrapper are both referenced explicitly. In the second case, the objects P and Q of the RefWrapper are referenced once using the new “ReferenceContents” property.

```
Wrapper(Test) {
    Reference = /RefWrapper/P;
    Reference = /RefWrapper/W;
}

Wrapper(Test) {
    ReferenceContents = /RefWrapper;
}

Wrapper(RefWrapper) {
    Property(P) {...}
    Wrapper(W) {...}
}
```

Related Topics

[Property](#)

[RepeatableProperty](#)

[DataWrapper](#)

[CsvWrapper](#)

[report_config_syntax](#)

Property

A wrapper used in metadata to define a property that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
    Property(property_name) {
        Usage      : normal | hidden | Env(EnvVar)
                    | AccessFeature(feature_name) ;
        InsertFirst ;
        InsertLast ;           // default
        InsertBefore : meta_leaf_name ;
        InsertAfter  : meta_leaf_name ;
        Options(Visualizer) {
            DisplayString   : string ;
            BlankSpaceAbove : on ;
        }
        Value(value_id) { // repeatable
        }
    }
}
```

Description

A wrapper used in metadata to define a property that is usable in configuration data.

A property exists as soon as its parent wrapper exists regardless of whether the property is specified or not.

Arguments

- Usage : normal | hidden | Env(EnvVar) | AccessFeature(*feature_name*) ;

A property that controls the visibility of the property. When set to normal, the property is reported by the [report_config_syntax](#) command and is visible in DFTVisualizer. When set to hidden, the wrapper is not reported by report_config_syntax and is invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands, such as the [get_config_value](#) command, if the name of the hidden property is known by the user. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden. When the Usage value is AccessFeature(*feature_name*), the usage value is “normal”, if the *access_code* for the *feature_name* is enabled, otherwise the Usage value is “hidden”.

- InsertFirst ;

An implicit Boolean property that specifies that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which properties are displayed in the Configuration Option panel inside DftVisualizer.

- InsertLast ;

An implicit Boolean property that specifies that the property is to be inserted after the already existing elements in the wrapper. This is useful if you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which properties are displayed in the Configuration Option panel in DftVisualizer.

- InsertBefore : *meta_leaf_name* ;

A property that specifies that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which properties are displayed in the Configuration Options panel in DftVisualizer.

- InsertAfter : *meta_leaf_name* ;

A property that specifies that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which properties are displayed in the Configuration Options panel in DFTVisualizer.

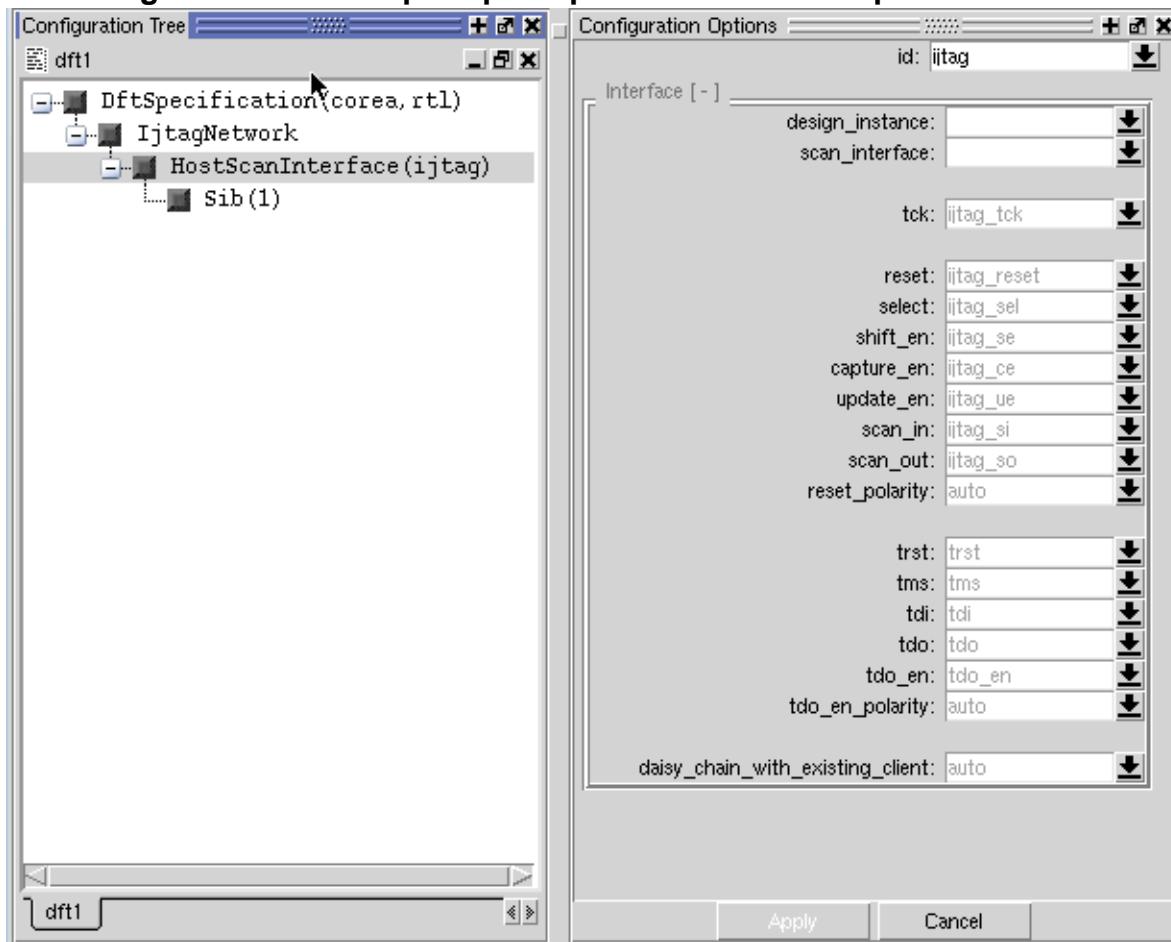
- Options(Visualizer)/DisplayString : *string* ;

A property used to control how the property is identified in the GUI. When unspecified, it defaults to *property_name*. This option is useful when you want to create a GUI to display configuration data and you want to name the properties using sentences as opposed to the strict property name.

- Options(Visualizer)/BlankSpaceAbove : On | Off ;

A property used to control the insertion of a blank space above the property when creating the Configuration Options panel. You can see its effect in the panel shown in [Figure 10-61](#) on page 3732. The option was specified inside the tck, reset, trst and daisy_chain_with_existing_client property wrappers.

Figure 10-61. Example Option panel where blank spaces were added



Examples

The following example defines properties of all types. It is used in the example of the `get_config_value` command to illustrate how to introspect them. You can use `report_config_data` DftSpecification -partition meta to see many more examples.

```
Wrapper(SingularWrapper) {
    Repeatable : single;
    Wrapper(WrapperWithRepeatableId) {
        id(name*) {
            type : string;
        }
        Reference = /SingularWrapper/WrapperWithRepeatableId;
        Property(time_property) {
            Value(time) {
                type : time;
                DefaultValue : 3.5ns;
            }
        }
        Property(integer_property) {
            Value(integer) {
                type : int;
                DefaultValue : 3;
            }
        }
        Property(binary_property) {
            Value(binary) {
                type : binary;
                DefaultValue : 3'b0;
            }
        }
        Property(string_property) {
            Value(string*) {
                type : string;
                DefaultValue : "";
            }
        }
    CSVWrapper(CsvWrapper) {
        Repeatable : on;
        id(module_name) {
            Type : String;
        }
        id(design_id) {
            Type : String;
            IgnoreCase : On;
        }
    }
    DataWrapper(DataWrapper_with_no_value) {
        RowIdElement(port) {
            Type : String;
            repeatable : on;
        }
    }
    DataWrapper(DataWrapper_value) {
        RowIdElement(port) {
            Type : String;
            repeatable : on;
        }
        Value(period) {
            Type : time;
        }
    }
}
}
```

Related Topics

[DftSpecification](#)

[RepeatableProperty](#)

[DataWrapper](#)

[CsvWrapper](#)

[report_config_syntax](#)

RepeatableProperty

A wrapper used in metadata to define a repeatable property that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
    RepeatableProperty(property_name) {
        Usage          : normal | hidden | Env(envVar)
                        | AccessFeature(feature_name) ;
        InsertFirst ;
        InsertLast ;           // default
        InsertBefore : meta_leaf_name ;
        InsertAfter  : meta_leaf_name ;
        Options(Visualizer) {
            DisplayString : string ;
        }
        Id(id) {             // repeatable and optional
        }
        Value(value_id) {   // repeatable, minimum one
        }
    }
}
```

Description

A wrapper used in metadata to define a repeatable property that is usable in configuration data.

As opposed to normal properties, a repeatable property has no default values and does not exist until it is specified.

Arguments

- Usage : normal | hidden | Env(EnvVar) | AccessFeature(*feature_name*) ;

A property that controls the visibility of a property. When set to normal, the property is reported by [report_config_syntax](#) and visible in DFTVisualizer. When set to hidden, the wrapper is not reported by [report_config_syntax](#) and invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands such as the [get_config_value](#) command if the name of the hidden property is known to the user. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden. When the Usage value is AccessFeature(*feature_name*), the usage value is “normal”, if the *access_code* for the *feature_name* is enabled, otherwise the Usage value is “hidden”.

- InsertFirst ;

An implicit Boolean property that specifies that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the properties are displayed in the Configuration Options panel in DFTVisualizer.

- InsertLast ;

An implicit Boolean property that specifies that the property is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the properties are displayed in the Configuration Options panel in DftVisualizer.

- InsertBefore : *meta_leaf_name* ;

A property that specifies that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the properties are displayed in the Configuration Options panel in DftVisualizer.

- InsertAfter : *meta_leaf_name* ;

A property that specifies that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the properties are displayed in the Configuration Options panel in DftVisualizer.

- Options(Visualizer)/DisplayString : *string* ;

A property used to control how the folding region associated with a repeatable property is identified in the Configuration Data window. When unspecified, it defaults to *property_name*. A convention of using lower case name with underscore is employed when naming normal and repeatable properties in the tool. The convention for the folding region name in DftVisualizer is to use Camel naming. This is achieved by using this option.

Examples

The following example shows how the connection property is defined inside the [Tdr](#) wrapper.

```
> report_config_data [get_config_value Tdr/DataOutPorts/Connection \
    -in_wrapper /DftSpecification/IjtagNetwork/HostScanInterface \
    -partition meta -object] RepeatableProperty(connection) {
Options(Visualizer) {
    DisplayString : Connections ;
}
Id(range) {
    Type : string ;
    Repeatable : On ;
}
Value(pin_name) {
    Type : string ;
    Repeatable : On ;
}
}
```

Related Topics

[DftSpecification](#)

[Property](#)

[DataWrapper](#)

[CsvWrapper](#)

[report_config_syntax](#)

DataWrapper

A wrapper used in metadata to define a data wrapper that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
    DataWrapper(wrapper_name) {
        Usage           : normal | hidden | Env(envVar)
                           | AccessFeature(feature_name) ;
        InsertFirst ;
        InsertLast ; //default
        InsertBefore   : meta_leaf_name ;
        InsertAfter    : meta_leaf_name ;
        Options(Visualizer) {
            DisplayString : string ;
        }
        RowIdElement(element_id) { // not_repeatable, required
        }
        Value(value_id) {          // repeatable, optional
        }
    }
}
```

Description

A wrapper used in metadata to define a data wrapper that is usable in configuration data.

As opposed to a normal wrapper, a data wrapper contains only data rows composed of arbitrary row id element with optional values. This type of wrapper is useful to associate values to arbitrary elements such as ports:

```
ClockPeriods {
    clka : 4ns;
    clkb : 3ns;
}
```

A data wrapper is like a wrapper with “Repeatable : Single”. It is not repeatable and starts to exist as soon as its parent wrapper exists.

Arguments

- Usage : normal | hidden | Env(EnvVar) | AccessFeature(*feature_name*) ;

A property that controls the visibility of the data wrapper. When set to normal, the data wrapper is reported by [report_config_syntax](#) and visible in DFTVisualizer. When set to hidden, the data wrapper is not reported by report_config_syntax and invisible in DFTVisualizer. Its content can still be fetched and edited using the configuration introspection and editing commands such as the [get_config_value](#) command if the name of the hidden data wrapper is known to the user. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1 otherwise it is treated as hidden. When the Usage value is AccessFeature(*feature_name*), the usage value is

“normal”, if the *access_code* for the *feature_name* is enabled, otherwise the Usage value is “hidden”.

- InsertFirst ;

An implicit Boolean property that specifies that the data wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside DFTVisualizer.

- InsertLast ;

An implicit Boolean property that specifies that the data wrapper is to be inserted after all already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command; it also controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside DFTVisualizer.

- InsertBefore : *meta_leaf_name* ;

A property that specifies that the data wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a data wrapper before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside DftVisualizer.

- InsertAfter : *meta_leaf_name* ;

A property that specifies that the data wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a data wrapper after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command. Also, it controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside DftVisualizer.

- Options(Visualizer)/DisplayString : *string* ;

A property used to control how the folding region associated with a data wrapper is identified in the Configuration Data window. When unspecified, it defaults to *wrapper_name*. This option is useful when you want to create a GUI to display configuration data and you want to name the folding regions as opposed to the strict data wrapper name.

Examples

The following example shows how the ClockPeriods data wrapper shown in the description section above is defined in metadata. Notice the value of type time with the extra allowed symbols sync and sync2x.

```
  DataWrapper(ClockPeriods) {
    RowIdElement(clock_port) {
      Repeatable : Off ;
      Type       : String ;
      IgnoreCase : Off ;
    }
    Value(period) {
      Type           : Time ;
      AllowedSymbols : sync, sync2x ;
    }
  }
```

Related Topics

[DftSpecification](#)

[Property](#)

[RepeatableProperty](#)

[CsvWrapper](#)

[report_config_syntax](#)

CsvWrapper

A wrapper used in metadata to define a CSV wrapper that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
    CsvWrapper(wrapper_name) {
        Usage      : normal | hidden | Env(envVar)
                    | AccessFeature(feature_name) ;
        InsertFirst ;
        InsertLast ;      // default
        InsertBefore : meta_leaf_name ;
        InsertAfter  : meta_leaf_name ;
        Id(id) {         // repeatable and optional
            }
    }
}
```

Description

A wrapper used in metadata to define a CSV wrapper that is usable in configuration data.

As opposed to a normal wrapper, a CSV wrapper contains only rows composed of arbitrary comma separated values.

```
csv {
    v1,v2,"v3 with spaces or comma";
    v4,v5;
}
```

See the command description of [report_config_data](#) and [write_config_data](#) to see how the wrapper content can be imported from or exported to Excel. CSV wrappers are not visible nor editable inside DftVisualizer. They are meant to be manipulated graphically inside Excel.

Arguments

- Usage : normal | hidden | Env(EnvVar) | AccessFeature(*feature_name*) ;

A property that controls the visibility of the CSV wrapper. When set to normal, the wrapper is reported by the [report_config_syntax](#) command. When set to hidden, the wrapper is not reported by report_config_syntax. Its value can still be fetched and edited using the configuration introspection and editing commands such as [get_config_value](#) if the name of the hidden CSV wrapper is known to the user. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden. When the Usage value is AccessFeature(*feature_name*), the usage value is “normal”, if the *access_code* for the *feature_name* is enabled, otherwise the Usage value is “hidden”.

- InsertFirst ;

An implicit Boolean property that specifies that the CSV wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra

metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command.

- **InsertLast ;**

An implicit Boolean property that specifies that the CSV wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command.

- **InsertBefore : *meta_leaf_name* ;**

A property that specifies that the CSV wrapper is to be inserted before the already existing elements in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a CSV wrapper before Wrapper(IjtagNetwork), you would specify “InsertBefore : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command.

- **InsertAfter : *meta_leaf_name* ;**

A property that specifies that the CSV wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its *leaf_name*. For example, to insert a CSV wrapper after Wrapper(IjtagNetwork), you would specify “InsertAfter : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the [report_config_syntax](#) command.

Examples

The following example shows how a generic scratch CSV wrapper is defined in the tool. You can use scratch CSV wrappers to import and introspect CSV data.

```
CsvWrapper(csv) {
    Repeatable : On ;
    Usage : hidden ;
    Id(name) {
        Type : string ;
    }
}
```

Related Topics

[DftSpecification](#)

[Property](#)

[RepeatableProperty](#)

[DataWrapper](#)

[report_config_syntax](#)

Id

A wrapper used in metadata to define an identifier for named wrappers, repeatable properties, and named CSV wrappers.

Syntax

```

Wrapper(wrapper_name) {
    Id (name_id) {
        Type : String | Boolean | Real | Int | Time | Binary ;
        LegalString : string_list ;           // For String
        LegalInt : integer_range_list ;      // For Int
        LegalReal : real_range_list ;        // For Real
        LegalSize : int | Unsigned ;         // For Binary
        AllowedSymbols : string_list ;       // For all but String
        IgnoreCase : On | Off ;
        Repeatable : On | Off ;
        Mandatory : On | Off ;
        Option(Visualiser) {
            Counter : local | global |
                        ParentWrapper(wrapper_name).var_name ;
            DefaultValueList : string_list ;
        }
    }
    RepeatableProperty(property_name) {
        Id (name_id) {
        }
    }
    CsvWrapper(wrapper_name) {
        Id (name_id) {
        }
    }
}

```

Description

A wrapper used in metadata to define an identifier for named wrappers, repeatable properties, and named CSV wrappers.

Parameters

- *name_id*

A string used to name the identifier. This name can later be referenced using the “[get_config_value -id name_id](#)” command.

The *name_id* must start with a letter, include only letters, numbers and underscores, and may be terminated by an asterisk “*”. If the *name_id* ends with an *, the Id is repeatable. If there is more than one Id wrapper, only the last one is allowed to have an * at the end. For example:

```

Wrapper(Step) {
    Id(instance*) {
    }
}

```

allows Step(u1,u2), where u1 and u2 are repeated instance Ids.

- **Type : value ;**

A property that defines the type of the Id. The allowed types are String, Boolean, Real, Int, Time, and Binary. Ids of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>'b<binary_digits>, 'b<binary_digits>, <size>'h<hexadecimal_digits>, and 'h<hexadecimal_digits>. The default selection is of type String.

- **LegalString : string_list ;**

A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed symbols is case insensitive by default or based on the Id's IgnoreCase property.

- **LegalInt : integer_range_list ;**

A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

Instead of using only numerical values, integer_range_list can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.

- **LegalReal : real_range_list ;**

A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- **LegalSize : int | Unsized ;**

A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- **AllowedSymbols : string_list ;**

A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols is case insensitive by default or based on the Id's IgnoreCase property.

- **IgnoreCase : On | Off ;**

A property that can be used to make the LegalString and the AllowedSymbols case sensitive. By default, the LegalString and the AllowedSymbols are case insensitive.

- Repeatable : On | Off ;

A property that controls if a given name _id can be repeated or not. The validation of the repeatability will consider casing or not based on the IgnoreCase property documented above.

- Mandatory : On | Off ;

A property that specifies if the Id is required or not. This option can only be specified Off if you have more than one Id wrapper, the Id wrapper is not the first one and all Id wrappers below it also have this property set to Off.

- Option(Visualiser)/Counter : local | global | ParentWrapper(wrapper_name).var_name ;

A property used to define how the Id will be populated automatically when adding the wrapper graphically inside DftVisualizer. When the DefaultValueList reaches the element with a # character, the # character is replaced by an integer. The integer is a counter that is either local to the wrapper, global to the entire root wrapper, or local to a named variable attached to a parent wrapper. For example, Ids defined with a counter as ParentWrapper(controller).memory_interface will share a counter when found anywhere below a wrapper called “controller”. The specified parent wrapper leaf name must exist in the ancestry of the element defining the Id.

- Option(Visualiser)/DefaultValueList : string_list ;

A property used to define a list of strings that will be used for the first, second, and so on instances of the wrapper. The string elements may contain any number of letters, numbers and underscores. The last element of the list must include a # character and may be just a # character. For example, the IJtagNetwork/HostScanInterface/Id(id) is defined with a DefaultValueList of “ijtag, ijtag#”. If you add a first HostScanInterface, you will see its Id will be “ijtag”. Any other subsequent instances will be labelled ijtag0, ijtag1, and so on.

Examples

The following example defines a wrapper with two Ids, the second one is repeatable and optional.

```
Wrapper(EX1) {
    Id(name) {
        type : string;
    }
    Id(period*) {
        type : time;
        Mandatory : off;
    }
}
```

Related Topics

[DftSpecification](#)

[RepeatableProperty](#)

[CsvWrapper](#)

Value

report_config_data

Value

A wrapper used in metadata to define an identifier for values of properties, repeatable properties, and data wrappers.

Usage

```
Wrapper(wrapper_name) {
    Property(property_name) {
        Value (value_id) {
            Type           : String | Boolean | Real | Int | Time | Binary |
                               ImplicitBoolean ;
            LegalString   : string_list ;           // For String
            LegalInt      : integer_range_list ;  // For Int
            LegalReal     : real_range_list ;      // For Real
            LegalSize     : int | Unsigned ;       // For Binary
            DefaultValue   : default_value ;
            AllowedSymbols: string_list ;         // For all but String and
                                               // ImplicitBoolean
            IgnoreCase     : On | Off ;
            Mandatory     : On | Off ;
            SizedBinary   : On | Off ;
        }
    }
    RepeatableProperty(property_name) {
        Value (value_id) {
    }
}
DataWrapper(wrapper_name) {
    Value (value_id) {
}
}
```

Description

A wrapper used in metadata to define an identifier for values of properties, repeatable properties, and data wrappers.

Arguments

- *value_id*

A string used to name the value. You can reference this name later by using the “[get_config_value -value_id value_id](#)” command.

The *value_id* must start with a letter, include only letters, numbers and underscores and may be terminated by an asterisk “*”. If the *value_id* ends with an *, a comma separated list of values can be entered for that value. If there is more than one Value wrapper, only the last one is allowed to have an * at the end.

Example 1:

```
RepeatableProperty(instance) {
    Value(instance_path) {}
}
```

- instance is specified as: instance : chip/module/instance;
- Result: value_id “instance_path” will contain “chip/module/instance”.

Example 2:

```
RepeatableProperty(instance) {
    Value(power_domain) {}
    Value(instance_path) {}
}
```

- instance is specified as: instance : domain1, chip/module/instance;
- Result: value_id “power_domain” will contain “domain1”
 value_id “instance_path” will contain “chip/module/instance”

Example 3:

```
RepeatableProperty(instance) {
    Value(power_domain) {}
    Value(instance_paths*) {}
}
```

- Where instance is specified as:
 instance : domain1, instance1, instance2, “chip/module/\escaped_inst”;
- Result: value_id “power_domain” will contain “domain1”
 value_id “instance_paths” will be returned as a Tcl list containing
 ‘instance1, instance2, {chip/module/\escaped_inst }’ by [get_config_value](#).

- Type : *value* ;

A property that defines the type of the Value. The allowed types are String, Boolean, Real, Int, Time, Binary, and ImplicitBoolean. Values of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>. ImplicitBoolean values are implied true as soon as the property is specified with no value. The property InsertFirst in the [Property](#) wrapper is an example of a property with Type ImplicitBoolean. The default selection is of type String.

- LegalString : *string_list* ;

A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list will consider casing or not based on the Value’s IgnoreCase property.

- LegalInt : *integer_range_list* ;

A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

Instead of using only numerical values, *integer_range_list* can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit

<p>integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.</p> <ul style="list-style-type: none"> • LegalReal : <i>real_range_list</i> ; A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”. • LegalSize : int Unsized ; A property that can be used when Type is set to Binary to limit the binary number to a specific size. • DefaultValue: <i>default_value</i> ; A property that is mandatory and only used when the Value wrapper is inside a Property wrapper. The specified <i>default_value</i> provide the value of the property when the property is not specified in the wrapper where it exist. • AllowedSymbols : <i>string_list</i> ; A property that can be used when Type is set to anything but String and ImplicitBoolean to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property for Value is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols will consider casing or not based on the Value’s IgnoreCase property. • IgnoreCase : <u>On</u> Off ; A property that can be used to make the LegalString and the AllowedSymbols case-insensitive. • Mandatory : On <u>Off</u> ; A property that specifies if the Value is required or not. This option can only be specified Off for second level or lower Value wrappers and if the Value wrappers below it are also Off. • SizedBinary : On <u>Off</u> ; A property that specifies that the binary value from the configuration data must be a sized binary. It must be a sized binary, hexadecimal, or decimal number using the <i>int'b</i>, <i>int'h</i>, or <i>int'd</i> prefix.

Examples

The following example defines a property with two values; the second one is repeatable and optional.

```
Property(EX1) {
    Value(name) {
        type : string ;
    }
    Value(period*) {
        type : time ;
        Mandatory : off ;
    }
}
```

Related Topics

[DftSpecification](#)

[RepeatableProperty](#)

[CsvWrapper](#)

[report_config_syntax](#)

RowIdElement

A wrapper used in metadata to define the syntax of the RowIdElements of a data wrapper.

Usage

```
  DataWrapper(wrapper_name) {
    RowIdElement(element_id) {
      Type : String | Boolean | Real | Int | Time | Binary ;
      LegalString : string_list ;           // For String
      LegalInt : integer_range_list ;      // For Int
      LegalReal : real_range_list ;        // For Real
      LegalSize : int | Unsigned ;         // For Binary
      AllowedSymbols : string_list ;       // For all but String
      IgnoreCase : On | Off ;
      Repeatable : On | Off ;
    }
  }
```

Description

A wrapper used in metadata to define the syntax of the RowIdElements of a data wrapper.

Arguments

- *element_id*

A string used to name the row id element.

The *element_id* must start with a letter, include only letters, numbers and underscores.

- Type : String | Boolean | Real | Int | Time | Binary ;

A property that defines the type of the RowIdElement. The allowed types are String, Boolean, Real, Int, Time, and Binary. RowIdElements of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>'b<binary_digits>, 'b<binary_digits>, <size>'h<hexadecimal_digits>, and 'h<hexadecimal_digits>.

- LegalString : *string_list* ;

A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list will consider casing or not based on the IgnoreCase property documented below.

- LegalInt : *integer_range_list* ;

A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

Instead of using only numerical values, *integer_range_list* can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.

- LegalReal : *real_range_list* ;
A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.
- LegalSize : *int* | Unsized ;
A property that can be used when Type is set to Binary to limit the binary number to a specific size.
- AllowedSymbols : *string_list* ;
A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property above is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols will consider casing or not based on the IgnoreCase property documented below.
- IgnoreCase : On | Off ;
A property that can be used to make the LegalString and the AllowedSymbols case-insensitive. This option also affects whether casing is ignored when performing the repeatability check described in the following Repeatable property.
- Repeatable : On | Off ;
A property that controls whether a given element_id can be repeated or not within the Data Wrapper. The validation of the repeatability will consider casing or not based on the preceding IgnoreCase property.

Examples

The following example defines a data wrapper with RowIdElement which is a arbitrary string meant to represent a clock port name. Each clock port name are only allowed once as specified by the fact that Repeatable is set to Off. Because IgnoreCase is also Off, port names Clk and clk will not be considered as repeated port names.

```
DataWrapper(ClockPeriods) {
    RowIdElement(clock_port) {
        Repeatable : Off ;
        Type       : String ;
        IgnoreCase : Off ;
    }
    Value(period) {
        Type           : Time ;
        AllowedSymbols : sync. sync2x ;
    }
}
```

Related Topics

[DftSpecification](#)

[RepeatableProperty](#)

[CsvWrapper](#)

[Value](#)

[report_config_syntax](#)

Chapter 11

Tessent Core Description

This section provides links to supporting documentation that describes the Tessent Core Description (TCD) configuration data syntax.

Refer to the links below for the primary documentation that describes the following TCD macro module types:

- **BoundaryScan Segments** — Refer to “[Tessent Core Description](#)” in the *Tessent BoundaryScan User’s Manual*.
- **FuseBoxInterface** — Refer to “[FuseBoxInterface](#)” in the *Tessent MemoryBIST User’s Manual*.
- **Memories** — Refer to “[Memory](#)” in the *Tessent MemoryBIST User’s Manual*.
- **Scan** — Refer to “[Scan](#)” on page 3756.
- **Shared Bus Memory Clusters** — Refer to “[MemoryCluster](#)” in the *Tessent MemoryBIST User’s Manual*.

[Scan](#).....[3756](#)

Scan

Describes the scan chains for the specific *module_name*.

Usage

```
Core(module_name) {
    Scan {
        module_type : normal | occ
        allow_scan_out_retiming : 1 | 0;
        is_hard_module : 1 | 0;
        exclude_from_concatenated_netlist : 1 | 0;
        pre_scan_drc_on_boundary_only : 1 | 0;
        internal_scan_only : 1 | 0;
        TestEn(port_name,...) {
            active_polarity : sized_binary | all_ones | all_zeros;
        }
        traceable : 1 | 0;
        ScanEn(scan_en_port_or_pin_name,...) {
            clock : port_name;
            active_polarity : sized_binary | all_ones | all_zeros;
        }
        Clock(clock_port_or_pin_name,...) {
            off_state : sized_binary | all_ones | all_zeros;
        }
        ClockOut(clock_out_port_or_pin_name,...) {
            slow_clock_input : port_name;
            fast_clock_input : port_name;
        }
        FastCaptureClockEn(pin_name_on_persistent_cell) {
            clock_input : port_name;
        }
        ShiftRegisterClockEn(pin_name_on_persistent_cell) {
            clock_input : port_name;
        }
        SetReset(set_reset_port_or_pin_name,...) {
            active_polarity : sized_binary | all_ones | all_zeros
        }
        ScanChain {
            length : int | auto;
            no_si_pipelineing : 1 | 0;
            scan_in_port : port_name,...;
            scan_out_port : port_name,...;
            scan_in_clock : port_name,...;
            scan_out_clock : port_name,...;
        }
        Iproc(proc_name) {
            parameter_value_list : parameter_value, ...;
        }
    }
}
```

Description

Describes the scan chains for the specific *module_name*. Such descriptions are automatically read during module matching. See the “[“set_design_sources -format tcd_scan” command](#)

description for information about where the tool searches for these items. See the [read_core_descriptions](#) command description for information on reading the descriptions explicitly. See the [set_module_matching_options](#) command description for information about the name matching process.

To see the content of a read-in Core(*module_name*)/Scan, use the following command syntax:

```
report_config_data Core(module_name)/Scan -partition tcd
```

To see the complete supported syntax, use the following command syntax:

```
report_config_syntax [get_config_value Core/Scan -partition meta:tcd -object]
```

Arguments

- **module_type** : [normal](#) | occ;

A property that defines the type of the module. Most module have type normal. The only special module is the OCC modules.

Identifying a module with type “occ” instructs the tool to infer a clock on the specified ClockOut location when entering the analysis mode in the dft -scan context if no clock exists. This ensures the scan insertion flow does not make clock connections for inserted retiming elements to a net that is not driven by the OCC. Also, the scan insertion uses this information to not align all the OCC chains, which may result in encoding conflicts when too many OCC care bits align.

- **allow_scan_out_retimings** : 1 | 0;

A Boolean property used to specify if the clock associated to scan chain is suitable for being used as the source for retiming elements or not. By default, when a segment needs retiming, the retiming element is added after the scan-out port using the clock port associated to the scan chain. In certain situations, this is not the best approach because the clock input is not balanced with the chain scan-out port. This situation happens in an OCC, for example, as can be seen in the [Schematic](#) diagram for the On-Chip Clock Controller in the *Tessent Scan and ATPG User's Manual*. When the property is set to 0, the retiming element is added at the scan-in port of the next scan element instead and the clock associated to the next scan element is used.

- **is_hard_module** : 1 | 0;

A property used to specify whether the module is hard or not. When set to 1, flip-flops not traced as part of the scan chains are assumed to be non-scannable. These scan chains are not collected by scan insertion and no DRC violation for those non scannable flops will be issued.

When set to 0, the module is assumed to be soft and the flip-flops not traced as part of the scan chains are assumed to be scannable unless explicitly declared non-scannable using the [add_nonscan_instances](#) command. The extra scannable flops may generate DRC violations, and if DRC clean, these flops are inserted in the newly-created scan chains.

- `exclude_from_concatenated_netlist : 1 | 0;`

A property used to specify that the module is a hard macro and should not be included in the concatenated netlist. A hard macro is a module that has its layout separate, but is too large to be a cell. You do not want to include these modules in the concatenated netlist because the synthesis and layout tool use the Liberty format instead of the netlist view. This property can only be set to 1 when the `is_hard_module` property is set to 1. Its value gets transferred to the `exclude_from_concatenated_netlist` module level attributes and affect the [write_design](#) command accordingly. See the documentation of the `-exclude_modules` switch of the [write_design](#) command description for more details.

- `pre_scan_drc_on_boundary_only : 1 | 0;`

A property used to specify that the module is to be treated as a black box. The scan chains are DRC'd from the external boundary of the module. They are not traced, and no DRC violation for the internals of the module are reported. This property is useful when the module requires a complex test_setup procedure to configure the module in scan mode.

For example, this feature is used to describe the chain going through the Tessent OCC controller. Because the scan chain length is configurable by the `max_capture_size` static signals, they would need to be configured to the maximum value for the tracing to work and max chain length considered during scan chain balancing. Instead of doing that, the maximum length of the chain is described using the `ScanChain/length` property. If the scan internals of the scan chain turned out to be have a design flaw, it would be detected right after scan insertion when performing DRC one scan mode at a time.

- `internal_scan_only : 1 | 0;`

A property used to identify that the scan chains of the module are allowed or not to be promoted to as being wrapper chains. See the [analyze_wrapper_cells](#) command description to understand how existing chain chains are handled by the analysis. When this property is set to 1, the scan chains are not allowed to be promoted to the shared wrapper chains and dedicated wrapper cells will be inferred to isolate them if the chain interacts with the outside.

- `TestEn(port_name,...)`

A wrapper used to define one or more TestEn ports on the module. See the description of the `ltest_en` DFT signal in [Table 3-10](#) on page 190 located in the [add_dft_signals](#) command description.

- `TestEn/active_polarity : sized_binary | all_ones | all_zeros;`

A property that defines the active polarity of the TestEn ports. The value must be a sized binary number where the width must match exactly the number of ports listed in the `TestEn` parenthesis. A “1” in the *n*th position means the *n*th TestEn port in the list is active high, and a “0” means it is active low. For example, the port list (`A, b[1:0]`), would require a 3-bit binary number where the left most bit corresponds to port `A` and the right most bit corresponds to bit `b[0]`. You can also use the symbols `all_ones` and `all_zeros` when they are all of the same active polarity.

- traceable : 1 | 0;

A property that is used to specify if the scan chain is traceable or not. When not traceable, the length, scan_in_clock and scan_out_clock properties inside the ScanChain wrapper must be specified as they cannot be extracted.

Note

Currently, it is not always possible to accurately identify the flops inside the OCC. Therefore, you are no longer allowed to change the traceable property.

- ScanEn(*scan_en_port_or_pin_name*,...)

A wrapper used to define one or more ports as having the function “ScanEn”. These ports are asserted active to place the scan chains in its shift configuration. At least one ScanEn port is required when there are one or more ScanChain wrappers.

- ScanEn/active_polarity : sized_binary | all_ones | all_zeros;

A property that defines the active polarity of the ScanEn ports. The value must be a sized binary number where the width must match exactly the number of ports listed in the ScanEn parenthesis. A “1” in the *n*th position means the *n*th ScanEn port in the list is active high, and a “0” means it is active low. For example, the port list (A, b[1:0]), would require a 3-bit binary number where the left most bit corresponds to port A and the right most bit corresponds to bit b[0]. You can also use the symbols all_ones and all_zeros when they are all of the same active polarity.

- Clock(*clock_port_or_pin_name*,...)

A wrapper used to define one or more ports as having the function “Clock”. All scan cells in the scan chain must be clocked by one of those defined Clock ports.

- Clock/off_state : sized_binary | all_ones | all_zeros;

A property that defines the offset of the clock ports when pulsing them to extract the scan chains. The value must be a sized binary number where the width must match exactly the number of ports listed in the ScanEn parenthesis. A “1” in the *n*th position means the *n*th ScanEn port in the list is active high, and a “0” means it is active low. For example, the port list (A, b[1:0]), would require a 3-bit binary number where the left most bit corresponds to port A and the right most bit corresponds to bit b[0].

The offstate is only relevant if your chains have a mixture of rising and falling edge flip-flops. Defining the off state incorrectly in that case may end up with the scan chain being extracted with a length that is not 100 percent accurate and may have a minor effect on chain balancing.

- ClockOut(*clock_out_port_or_pin_name*,...)

This wrapper defines one port or pin inside the module on which to add a clock when going to analysis mode in the dft -scan context. The ClockOut ports are only allowed and used when module_type is set to occ. See the description of the module_type property above for more information. The ClockOut port wrapper must include one slow_clock_port and/or fast_clock_port.

- ClockOut/slow_clock_input : *port_name*;

A string property use to provide the name of the input port through which the slow clock enters the module. This clock is used as the shift clock and optionally the capture when not operating in fast capture mode. If you have an OCC that only gates the clock and the shift clock is injected earlier in the clock tree, you will only have a fast_clock_port. The mini tck OCC present in the Sib(sti) only has a slow_clock_port.

- ClockOut/fast_clock_input : *port_name*;

A string property use to provide the name of the input port through which the fast clock enters the module. This clock is used as the capture clock when operating in fast capture mode. If you have an OCC that only gates the clock and the shift clock is injected earlier in the clock tree, you will only have a fast_clock_port.

- FastCaptureClockEn(*pin_name_on_persistent_cell*)

This wrapper is only used within the Core/Scan wrapper with a module_type equal to “occ”. The wrapper specifies the enable pin of the clock gating cell that gates the fast clock during logic test mode. This pin is forced off when running the DFT_C1, DFT_C2, DFT_C3, and DFT_C6 DRCs in order to check the functional clock source.

- FastCaptureClockEn/clock_input : *port_name*;

The property is currently unused.

- ShiftRegisterClockEn(*pin_name_on_persistent_cell*)

This wrapper is only used within the Core/Scan wrapper with module_type equal to “occ”. The wrapper specifies the enable pin of the clock gating cell that gates the clock of the shift register inside the OCC. This pin is forced off when running the DFT_C1, DFT_C2, DFT_C3, and DFT_C6 DRCs in order to check the functional clock source.

- ShiftRegisterClockEn/clock_input : *port_name*;

The property is currently unused.

- SetReset(*set_reset_port_name*,...)

A wrapper used to define one or more ports as having the function “SetReset”. These ports controls the Set and Reset pins of the scannable flip-flops found inside the module and are asserted inactive when tracing the scan chains.

- SetReset/active_polarity : sized_binary | all_ones | all_zeros;

A property that defines the active polarity of the SetReset ports. The value must be a sized binary number where the width must match exactly the number of ports listed in the ScanEn parenthesis. A “1” in the *n*th position means the *n*th SetReset port in the list is active high, and a “0” means it is active low. For example, the port list (A, b[1:0]), would requires a 3 bit binary number where the left most bit corresponds to port A and the right most bit corresponds to bit b[0]. You can also use the symbols all_ones and all_zeros when they are all of the same active polarity.

- ScanChain/length : int | auto;

A property used to define the length of the scan chain. The value auto is only allowed when the traceable property is 1. When the value is an integer and traceable is 1, a warning is generated if the traced length differs from the specified length and the traced length is used. The minimum scan chain length is one scan cell.

- ScanChain/no_si_pipelining : 1 | 0;

A Boolean property used to specify if the scan chain allows the insertion of a pipelining flip-flop in front of the scan_in_port. When set to 1, the no_si_pipelining property overrides any pipelining instructions set by the [set_scan_insertion_options -si_pipelining](#) command specified with “all_chains” or “wrapper_chains_only”.

Scan chains defined inside tcd_scan wrappers and having module_type equal to occ have no_si_pipelining inferred by the tool to 1. The tcd_scan data created by scan insertion automatically sets this property to 1 if the scan chain starts with a scan element having the no_si_pipelining property set or inferred to 1, or when a pipelining flip-flop is already inserted inside the core.

When set to 0, the tool does not prevent the insertion of a pipelining flip-flop on the scan chain in front of the scan_in_port and follows the direction set by the [set_scan_insertion_options -si_pipelining](#) command.

- ScanChain/*scan_in_port* : *port_name*, ...;

A property used to define the name of the scan-in port for the chain.

- ScanChain/*scan_out_port* : *port_name*, ...;

A property used to define the name of the scan-out port for the chain.

- ScanChain/*scan_in_clock* : *port_name*, ...;

A property used to defined the name of the clock used by the first scan flop of the chain. This property is optional when traceable is 1. A warning is generated if the clock of the first scan element of the chain is extracted and found to be different than the specified one. The extracted one is used when they differ.

- ScanChain/*scan_out_clock* : *port_name*, ...;

A property used to defined the name of the clock used by the last scan flop of the chain. This property is optional when traceable is 1. A warning is generated if the clock of the last scan element of the chain is extracted and found to be different than the specified one. The extracted one is used when they differ.

- Iproc(*proc_name*)

This wrapper defines the name of an iProc that must be used to activate the OCC. This iProc is automatically iCalled from the parent level when the OCC is still active in the external test modeof the child. If you are using the Tessent OCC, this is automatically populated with the name “setup”.

- `Iproc(proc_name)/parameter_value_list : parameter_value, ... ;`

This wrapper defines the parameters to used when calling the iProc mentioned in the parent wrapper. If you are using the Tessent OCC, this is automatically populated with this parameter value pair: “`fast_capture_mode, 1`”.

Related Topics

[add_scan_chains](#)

[read_core_descriptions](#)

[set_design_sources](#)

[set_module_matching_options](#)

Chapter 12

Tessent Shell Data Base (TSDB)

This chapter describes the Tessent Shell Database which is a structured directory containing subdirectories and files. The subdirectory naming is designed to group related information for each specific view of each design block. The complete subdirectory is deleted and recreated cleanly when a step is repeated thereby ensuring that no stale files from a previous run can remain in the active TSDB directory.

The [root_directory](#) name can be anything and located anywhere, and is specified using the [set_tsdb_output_directory](#) command. You can use a single root directory as the output of all of your design blocks but you can use as many as you like. As long as you make the TSDB directories associated to child blocks visible (using the [open_tsdb](#) command) to the tool when processing a given block, it does not matter if you have everything consolidated into one directory or distributed into many.

root_directory	3763
instruments	3765
dft_inserted_designs	3768
patterns	3774
logic_test_cores	3776
<code><design_name>.lbist_mode[_<mode_name>]</code>	3777
<code><design_name>.atpg_mode[_<mode_name>]</code>	3778
<code><design_name>.atpg_retargeting_mode[_<mode_name>]</code>	3779

[root_directory](#)

A directory that groups up to four subdirectories containing related information.

Summary

Figure 12-1. Root Directory Structure



Description

A directory that groups up to four subdirectories containing related information. The **root_directory** is specified using the [set_tsdb_output_directory](#) command. The subdirectories are created by the first command that needs to create a subdirectories inside them.

- **instruments** — This directory is the first time you issue the [process_dft_specification](#) or the “[write_edt_files -tsdb](#)” command and switch.
- **dft_inserted_designs** — This directory is created when you do either of the following:
 - The first time you issue the [process_dft_specification](#) command without specifying the [-no_insertion](#) option.
 - The first time you issue the “[extract_icl -write_in_tsdb](#)” command and switch.
- **patterns** — This directory is created the first time you issue the [process_patterns_specification](#) command.
- **logic_test_cores** — This directory is created the first time you issue the [write_tsdb_data](#) command in the patterns -scan context (see “[Contexts and System Modes](#)” in the *Tessent Shell User’s Manual*). Currently, only the LogicBIST patterns are supported by the [write_tsdb_data](#) command.

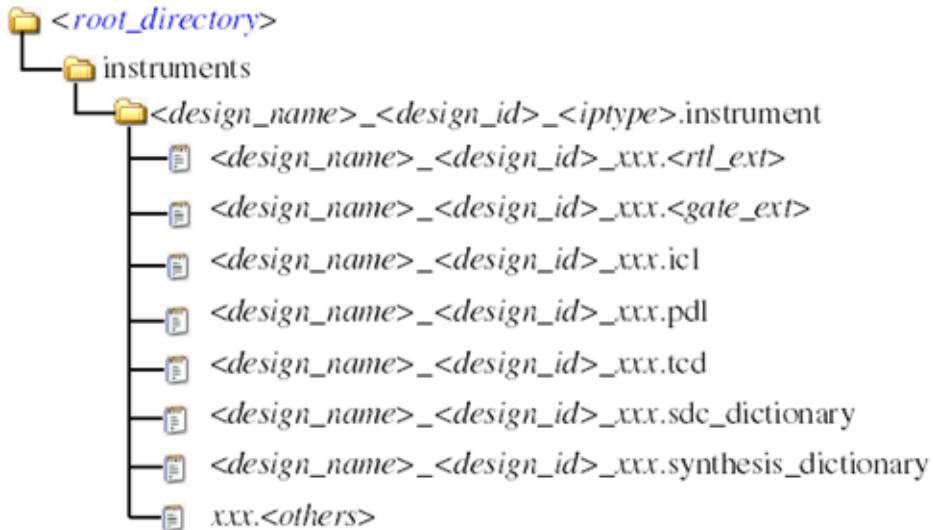
Components

The different subdirectories are documented in their respective sections in this chapter.

instruments

An instruments directory that contains a series of subdirectories that are referred to as “instrument containers”.

Summary



Description

The instruments directory contains a series of subdirectories that are referred to as “instrument containers”. The instrument container names are formed by concatenating the name of the design, the design identifier, and a unique string that is specific to the instrument or IP type. [Table 12-1](#) shows the string used for the various instrument types supported by the tool. See the “[set_context -design_identifier](#)” command to learn more about the design identifier string.

You can only have one instrument container per design_name/design_id/ ip_type combination visible in the opened TSDBs. If you have multiple TSDB directories containing the output of multiple runs on the same design with the same design id, and you try to open a second TSDB using the [open_tsdb](#) command, the tool will report an error if the new TSDB exposes a second copy of the same instrument container. If this occurs, you need to close the first TSDB before opening the second one. Also note that the current TSDB output directory, specified with the [set_tsdb_output_directory](#) command, is always open.

Table 12-1. Instrument Container Names

Instrument Type	Instrument container name
Boundary Scan	<design_name>_<design_id>_bscan.instrument
EDT	<design_name>_<design_id>_edt.instrument

Table 12-1. Instrument Container Names (cont.)

Instrument Type	Instrument container name
Hybrid EDT/LBIST	<design_name>_<design_id>_edt_lbist.instrument
IJTAG Network	<design_name>_<design_id>_ijtag.instrument
Logic Bist	<design_name>_<design_id>_lbist.instrument
LPCT	<design_name>_<design_id>_edt.instrument
Memory BISR	<design_name>_<design_id>_mbisr.instrument
Memory BIST	<design_name>_<design_id>_mbist.instrument
ncp_index_decoder	<design_name>_<design_id>_lbist_ncp_index_decoder.instrument
OCC	<design_name>_<design_id>_occ.instrument
RTL cells	<design_name>_<design_id>_cells.instrument

Components

- <design_name>_<design_id>_xxx.<rtl_ext>
One or many files containing the Verilog RTL description of the instrument. The extension defaults to “v” but is user-specified using the rtl_extension property in the DFTSpecification.
- <design_name>_<design_id>_xxx.<gate_ext>
One or many files containing the Verilog netlist description of the instrument. The extension defaults to “vg” but is user-specified using the gate_extension property in the DFTSpecification. This view of the instrument is only generated when synthesizing the instrument using the run_synthesis command which is only used when the -no_rtl insertion flow is used. In the -rtl insertion flow, the instruments are synthesized in context along with the rest of the functional logic.
- <design_name>_<design_id>_xxx.icl
One or many files containing the ICL descriptions of the instruments. The extension is always “icl”. These ICL descriptions are collected into a unified ICL file and stored in a [dft_inserted_designs](#) subdirectory when running ICL extraction.
- <design_name>_<design_id>_xxx.pdl
One or many files containing the PDL iProcs for the instruments. The extension is always “pdl”. The PDL files contain an iProcsForModule statement referring to an ICL module name followed by one or many iProcs. The PDL files are concatenated into a single file and stored beside the unified ICL file when running ICL extraction.
- <design_name>_<design_id>_xxx.tcd

One or many files containing extra information about the instrument. The “tcd” extension stands for “Tessent Core Description”. This file uses the configuration data format to encode extra information needed while using the instrument. For example, memory BIST uses this file to store the complete information about the memories tested by the controller such that precise diagnosis of failures can be done later. The TCD configuration file consists of a wrapper Core(<module_name>) with a sub-wrapper specific to the instrument type. Again, for example, memory BIST creates a “Core(<ctrl_mod_name>)/MemoryBistController” to store all of the information needed to diagnose failures on memories tested by a given controller.

- <design_name>_<design_id>_xxx.sdc_dictionary

One or many files containing a Tcl dictionary capturing timing exceptions associated with the instrument modules. The instance-specific information ends up in the extracted ICL file. The true SDC procs ends up being created in the *dft_inserted_designs* subdirectory when running ICL extraction (using the [extract_icl](#) or [set_system_mode](#) commands in analysis mode).

- <design_name>_<design_id>_xxx.synthesis_dictionary

One or many files containing a Tcl dictionary capturing tool-independent synthesis instructions. The information is used by the [run_synthesis](#) command to understand which modules need to be synthesized, which RTL files to read, and which netlist files to write out. This file is created systematically but is typically only created when using the -no_rtl insertion flow. When using the rtl insertion flow, the [run_synthesis](#) is not used and the instruments are synthesized in context along with the rest of the functional logic.

- xxx.<others>

Optional extra files having custom extensions specific to each instrument type that enable features that are specific to each of them. For example, the boundary scan instrument creates a BSDL file for the current design with the extension “.bsdl” to describe the boundary scan implementation.

Related Topics

[DftSpecification](#)

[PatternsSpecification](#)

[process_dft_specification](#)

[write_edt_files](#)

[process_patterns_specification](#)

[extract_icl](#)

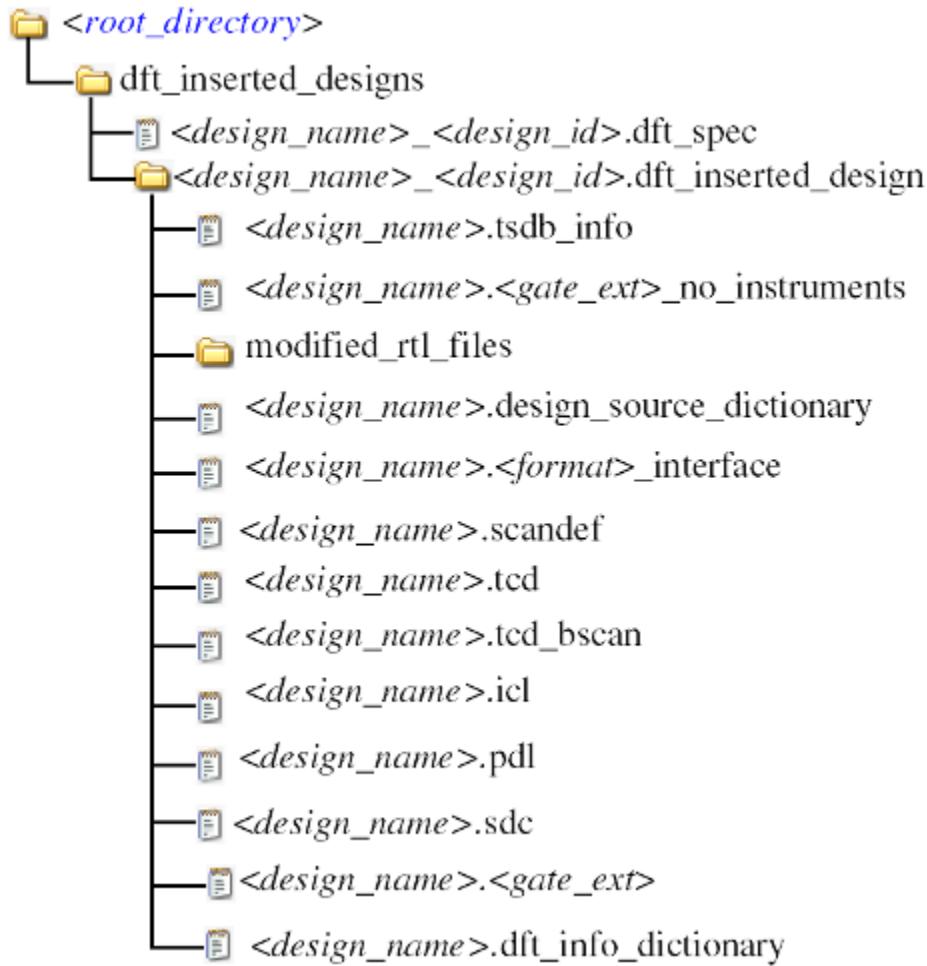
[run_testbench_simulations](#)

[run_synthesis](#)

dft_inserted_designs

A directory that contains one or many subdirectories that capture the information relative to a design in which some DFT has been inserted.

Summary



Description

The `dft_inserted_designs` directory contains one or many subdirectories capturing the information relative to a design in which some DFT has been inserted. The name of the subdirectory is formed by concatenating the name of the design with the design identifier followed by the “`.dft_inserted_design`” extension. See the [set_context -design_identifier](#) option description to read about the design identifier string value.

The directory is created and populated when the [process_dft_specification](#) command runs unless the `-no_insertion` option is specified. The directory can also be created by the

“[write_edt_files -tsdb](#)” commands. Finally, if it was not already created by the [process_dft_specification](#) or the [write_edt_files](#) commands, the directory can be created by the [extract_icl](#) command. The creating step is stored in the *.tsdb_info* file such that the tool knows to delete the directory completely before recreating it using the step that originally created it. This ensures there can never be any stale information in the *.dft_inserted_design* directory as you repeat the process.

You can only have one *.dft_inserted_design* directory per *design_name/design_id* combination visible in the opened TSDBs. If you have multiple TSDB directories containing the output of multiple runs on the same design with the same design id, and you try to open a second TSDB using the [open_tsdb](#) command, the tool will issue an error if the new TSDB exposes a second copy of the same *.dft_inserted_design* directory. If this occurs, you need to close the first TSDB before opening the second one. Also note that the current TSDB output directory, specified with the [set_tsdb_output_directory](#) command, is always open.

Components

- *<design_name>.tsdb_info*

A file that is created when the *.dft_inserted_design* directory is created to store information about the design view and the creation step. The format of this file is shown below.

```
TsdbInfo(module_name,design_id) {
    creation_date      : date;
    creation_user      : user_id;
    creation_step      : process_dft_specification | extract_icl |
                         write_edt_files;
    level              : chip | physical_block | sub_block;
    icl_extraction_needed : on | off;
    library_name       : logical_library_name;
    OpenedTsdbDirectories {
        path_to_tsdb_directory ; // repeatable
    }
    ChildBlockModules {
        module_name : design_id;
    }
}
```

You can read this file into memory using the [read_config_data](#) and introspect its content using the [get_config_value](#) command.

This file may grow over time as more information may be needed to automate new flows.

This file is created when the [process_dft_specification](#) runs unless the *-no_insertion* option is specified. It can also be created by the “[write_edt_files -tsdb](#)” commands. Finally, the file can be created by the [extract_icl](#) command, if it was not already created by the [process_dft_specification](#) or the [write_edt_files](#) commands.

- *<design_name>.<gate_ext>_no_instruments*

A Verilog netlist file, created when you use the “[set_context -no_rtl](#)” flow, that contains all of the design files forming the current physical block. The netlist of sub-blocks are included in this netlist but the netlist from child physical blocks is not included. See the [set_design_level](#) command for more information about the block types.

Note, this concatenated netlist does not include the netlist of the instruments inserted in the current insertion pass because they need to be synthesized prior to being included in the netlist.

This file is created by `process_dft_specification` when the `-no_insertion` option is not used, or by the “[write_edt_files -tsdb](#)” command.

- **modified_rtl_files**

A directory, created when using the “[set_context -rtl](#)” flow, that contains the RTL files that were edited during the current insertion pass. In the RTL flow, the file structure of the original RTL must be preserved. Because RTL supports five Verilog formats and four VHDL formats, they cannot be mixed into a common file because each file needs to be read with a unique [read_verilog](#) -format or [read_vhdl](#) -format command. Also, because of compiler directives that persist from one file to another, the file compilation order must be preserved to ensure that the proper design view is loaded. The list of modified files is taken and their common ancestor directory is computed. The relative path between every file and the common ancestor directory is preserved in the `modified_rtl_files` directory. For example, assume the following three files were modified: /a/b/c/f1, /a/b/c/f2, and /a/b/d/f3. The common ancestor is “/a/b”. The subdirectory “c” will be created in the `modified_rtl_files` directory to store f1 and f2, and the subdirectory “d” will be created in the `modified_rtl_files` directory to store f3. If included files are edited, their file names are uniquely modified and stored in the subdirectory called “INCLUDE”.

This directory is created by the `process_dft_specification` command when the `-no_insertion` option is not used

- **<design_name>.design_source_dictionary**

A file that contains a Tcl dictionary describing precisely how to load the current design. The directory consists of a series of ordered Read# entries containing the complete information to reproduce it. The information is general enough to be translated into read commands for any tool. The [run_testbench_simulations](#) command uses that information to create the compilation script for the supported simulators.

This file is created by the `process_dft_specification` command when the `-no_insertion` option is not used, or by `extract_icl` if not already created by the `process_dft_specification` command.

The format of the Read# entries is shown below:

```

read# { // read1 for the first read, read2 for the second, etc.
    format { // See read\_verilog/read\_vhdl -format option
        "verilog_1995" | "verilog_2001" | "verilog_sv31a" |
        "verilog_sv2005" | "verilog_sv2009" | "vhdl_1987" |
        "vhdl_1993" | "vhdl_2002" | "vhdl_2008"
    }
    work_library { //See read\_verilog/read\_vhdl -in_library option
        "<library_name>"
    }
    files {
        "<relative_file_name>"
        ...
    }
    libraries { // See set\_design\_sources command
        {directory | file <relative_directory_or_file_name>}
    }
    extensions { // See set\_design\_sources command
        <extension list used for searching in directory libraries>
    }
    defines { // See set\_design\_macros command
        "<macro_name>[=<value>]
        ...
    }
    incdirs { // See set\_design\_include\_directories command
        <relative_directory_name>
        ...
    }
    options { // See allowed options in read\_verilog and read\_vhdl
        // commands
        <option_used_in_read_command>
        ...
    }
}

```

This file is not generated when a design is assembled in -no_rtl context.

- <*design_name*>.<*format*>_interface

A file that contains only the port and parameter declarations of the current design. The format of the file is encoded in the extension such that the right format can be used to load it back. The table below shows the mapping between the extension and the format.

This file is created by the [process_dft_specification](#) command when the -no_insertion option is not used, or by [extract_icl](#) if it is not already created by [process_dft_specification](#) command.

Table 12-2. Format to extension mapping for the interface view

Format	extension
verilog 1995	v95_interface
verilog 2001	v_interface

Table 12-2. Format to extension mapping for the interface view (cont.)

Format	extension
verilog sv31a	sv31a_interface
verilog sv2005	sv05_interface
verilog sv2009	sv09_interface
vhdl 1987	vhd87_interface
vhdl 1993	vhd_interface
vhdl 2002	vhd02_interface
vhdl 2008	vhd08_interface

- *<design_name>.scandef*

A file that is created by [insert_test_logic](#) -write_in_tsdb on. It contains the scandef description of the scan chains so that you can perform scan chain reordering and re-partitioning during layout.

- *<design_name>.tcd*

A file that is created by the [process_dft_specification](#), [insert_test_logic](#) -write_in_tsdb on, and [write_design](#) -tsdb commands. It contains the DesignInfo and optionally the Scan wrappers within the Core(*design_name*) wrapper. The DesignInfo includes lots of data specified or created in an insertion pass so that it can be restored at the beginning of a future insertion pass. The file is always read-in when using the [read_design](#) command no matter which view is loaded ensuring that the DesignInfo is imported. The Scan wrapper describes the scan chains that were created by the [insert_test_logic](#) command and is reused in a hierarchical scan insertion process.

The information stored in the Scan wrapper is also used by the [import_scan_mode](#) command to import the clock, chain, and load_unload definitions of the scan mode such that it can be used to create ATPG modes.

- *<design_name>.tcd_bscan*

A file that is created by the [process_dft_specification](#) command when inserting a Boundary Scan chain into a sub or physical block and the block contains a Core(<module_name>)/BoundaryScan wrapper. This file is reused at the next level up to stitch the boundary scan chain in the parent module and eventually generate the BSDL file once the chip level is reached. See the [set_design_level](#) command for a description of the design levels.

- *<design_name>.icl*

A file that contains the concatenated ICL representation of the current design. It is generated when running the [extract_icl](#) command.

- *<design_name>.pdl*

A file that contains the content of all PDL files that were seen beside the input ICL and tcd_memory_lib files when running the [extract_icl](#) command. For every read-in ICL or tcd_memory_lib file, a PDL file having the same name and location but with the .pdl extension are loaded and concatenated into the merged PDL file. These PDL files must only contain a series of iProcsForModules and iProcs statements. The automatic collection of PDL files makes it easy to place a PDL file beside any ICL file and have it automatically collected during ICL extraction and automatically loaded after ICL elaboration so that it can be used in a [ProcedureStep](#) wrapper or with the [set_test_setup_icall](#) command.

The source of each imported file is marked with a comment specifying where it was taken from. Because ICL extraction is done hierarchically, a merged PDL file may be merged into a higher level PDL file. You will notice that the markers get extra # as they get included into a parent file as shown here:

```
# Imported from ../corea_rtl.dft_inserted_design/corea.pdl
## Imported from ../../../../../my_instrument/counter.pdl

The path "../../../../../my_instrument/counter.pdl" is relative to its
original file location "../corea_rtl.dft_inserted_design".
```

- <design_name>.sdc

A file that is created during ICL extraction or explicitly created using the [extract_sdc](#) command. The file combines the module-level timing information found in the .sdc_dictionary files with the instance-level information found in the ICL file. Refer to the “[Timing Constraints SDC](#)” in the *Tessent Shell User’s Manual* for complete information.

- <design_name>.<gate_ext>

A file that is typically created by the [run_synthesis](#) command after you have run the [process_dft_specification](#) command in the -no_rtl context. The [run_synthesis](#) command concatenates the <design_name>.<gate_ext>_no_instruments netlist with the newly-synthesized instruments found in the files at:

[instruments](#)/`<design_name>_<design_id>_xxx.<gate_ext>`

When using the [write_design](#) -tsdb command in the -no_rtl context, this file is directly created with the content of the current design found in memory. When using the [write_design](#) -tsdb -softlink_netlist command, the <design_name>.<gate_ext> entry is created as softlink to point to the location of the file containing the current design module in memory.

- <design_name>.dft_info_dictionary

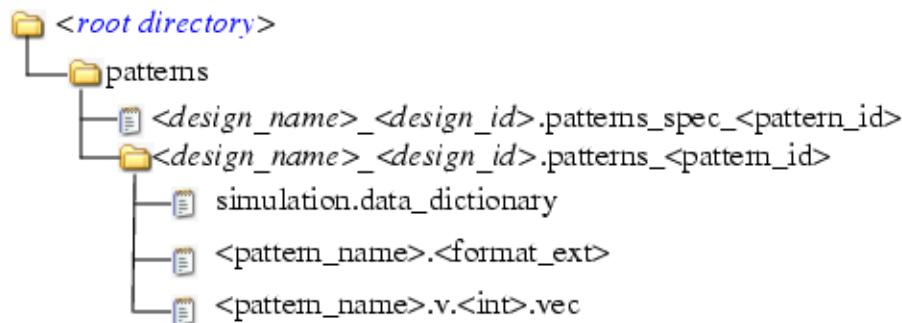
A file that is created during ICL extraction when using the [extract_icl](#) command with the -write_in_tsdb switch not set to off. The file is a Tcl dictionary containing the information about the DFT inserted in the design that must be considered during scan

insertion when using a third-party tool. See the [get_dft_info_dictionary](#) command for more information.

patterns

A directory that stores the patterns and pattern specifications associated with each design, design id, and pattern id.

Summary



Description

A directory used to store the patterns and patterns specifications associated with each design, design id, and pattern id. The patterns specification file is always updated upon each successful validation of the specification using the `process_patterns_specification` where is the TSDB output directory as specified using the [set_tsdb_output_directory](#) command.

Components

- <*design_name*>_<*design_id*>.patterns_spec_<*pattern_id*>
A file that contains the patterns specification that was just created using the [create_patterns_specification](#), and successfully validated using the [process_patterns_specification](#) command. The <*design_name*>, <*design_id*>, and <*pattern_id*> values correspond exactly to the matching elements in the PatternsSpecification(<*design_name*>, <*design_id*>, <*pattern_id*>) wrapper.
- <*design_name*>_<*design_id*>.patterns_<*pattern_id*>
A directory that is created the first time a given PatternsSpecification(<*design_name*>, <*design_id*>, <*pattern_id*>) wrapper is processed using the [process_patterns_specification](#) command. The <*design_name*>, <*design_id*> and <*pattern_id*> values correspond exactly to the matching elements in the PatternsSpecification(<*design_name*>, <*design_id*>, <*pattern_id*>) wrapper.

- simulation.data_dictionary

A file that contains a Tcl dictionary that is used by the [run_testbench_simulations](#) command to know the patterns list with their associated information as well as the design source dictionary explaining how to load the complete design downward. The format of the dictionary is illustrated below. Notice that the list of entries found in the Patterns entry is always the full list of patterns found in the PatternsSpecification wrapper independent of the -select or the -exclude option used in the last invocation of the process_patterns_specification command.

```
set simulation_dict {
    top_design_logical_library <library_name>
    patterns {
        <pattern_name> {
            testbench_module_name <module_name>
            top_design_instance_name <instance_name>
            extra_simulation_file_list <file_list>
            top_module_name_list <module_name_list>
            simulation_macro_list {<macro_definition_list>}
            lower_instances {
                <instance_name> {
                    view full | interface
                    logical_library <library_name>
                }
            }
            file_list {<testbench_and_sidefiles_list>}
            icl_time "<datetime>"
            sim_data_files <status>
        }
        design_source_dictionary {
            read# {
            }
        }
    }
}
```

- <pattern_name>.<format_ext>

A file that contains the top-level test bench or the patterns file expressed in the requested format. The extension of the file is “v” for Verilog test benches, or the exact format specified using the “manufacturing_patterns_format” property in the PatternsSpecification wrapper. If the “compress_pattern_files” option also found in the PatternsSpecification wrapper is specified to “on”, the patterns file are compressed and the .gz string is appended to the <format_ext> string.

Not all patterns found in the PatternsSpecification wrapper will be updated or created when running the process_patterns_specification command with the -select or the -exclude options. Stale pattern files left over in the <design_name>_<design_id>.patterns_<pattern_id> directories from a previous run of the process_patterns_specification command will be automatically deleted from the directory if they are no longer found in the current PatternsSpecification wrapper.

- <pattern_name>.v.<int>.vec

A set of files that is only created for Verilog test benches that contain the vector data for the associated pattern.

- <pattern_name>_<others>.v

A set of files that is only created for Verilog test benches that contain side modules to be compiled in parallel with the Verilog test bench. Such modules may include clock monitoring behavioral code. They may also contain bisr emulation code for sub-physical instances using the interface view. See the “emulate_bisr_chains_in_lower_physical_blocks” property inside the SimulationOptions wrapper for more information about those files.

logic_test_cores

A directory that stores the information generated when a test mode on a logic core was successfully design rule checked, and for which ATPG or LogicBIST patterns may have to be created.

Summary



Description

A directory used to store the information generated when a test mode on a logic core was successfully design rule checked, and for which ATPG or LogicBIST patterns may have been created.

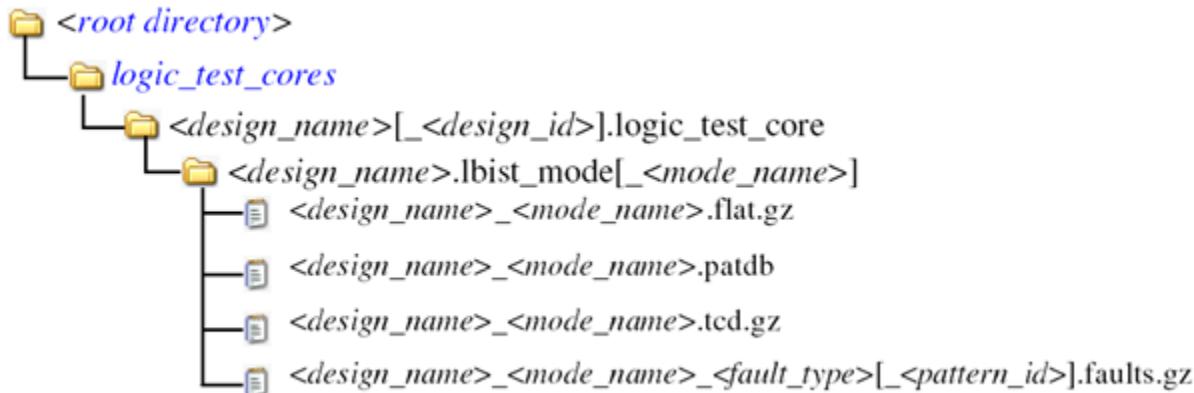
Components

- **<design_name>[_<design_id>].logic_test_core**
A directory used to hold all logic_bist, atpg, and atpg_retargeting containers. The directory uses the design name as its name with an optional design_id suffix that you can use to differentiate various views of the same design such as the pre-layout and post-layout versions.

<design_name>.lbist_mode[<mode_name>]

A directory used to store the information generated when a LogicBIST test mode on a logic core was successfully design rule checked and for which LogicBIST patterns were created.

Summary



Description

A directory used to store the information generated when a LogicBIST test mode on a logic core was successfully design rule checked and for which LogicBIST patterns were created.

Components

- **<design_name>.lbist_mode[<mode_name>]**
A directory used to store the information for a given LogicBIST test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the [set_context](#) command. The mode name is specified using the [set_current_mode](#) command.

The directory is created clean each time the [write_tsdb_data](#) command is issued in analysis mode and patterns -scan context. The mode is identified as a LogicBIST mode when there was at least one [add_lfsrs](#) command specified in setup mode prior to going to

analysis mode. You can use the “[get_context](#) -logic_bist” command to verify that it is a LogicBIST mode.

- <design_name>_<mode_name>.flat.gz

A file containing the flat model representing the DRCed LogicBIST test mode. This file is used to fault simulate other fault models at a later time without having to redo DRC. Its main utility is, however, to perform diagnosis when the design is in production. See the [Tessent Diagnosis User's Manual](#) for further information about how to run diagnosis and how it uses the flat model.

- <design_name>_<mode_name>.patdb

A file containing the LogicBIST pattern information. It contains all PRPG seeds, all MISR signatures, and the flop expect data for a specified amount of patterns.

- <design_name>_<mode_name>.tcd.gz

A file containing a Core/LbistMode wrapper describing different aspect of the logictest test mode. The stored information is mainly used to map diagnostic data back to flip-flop names.

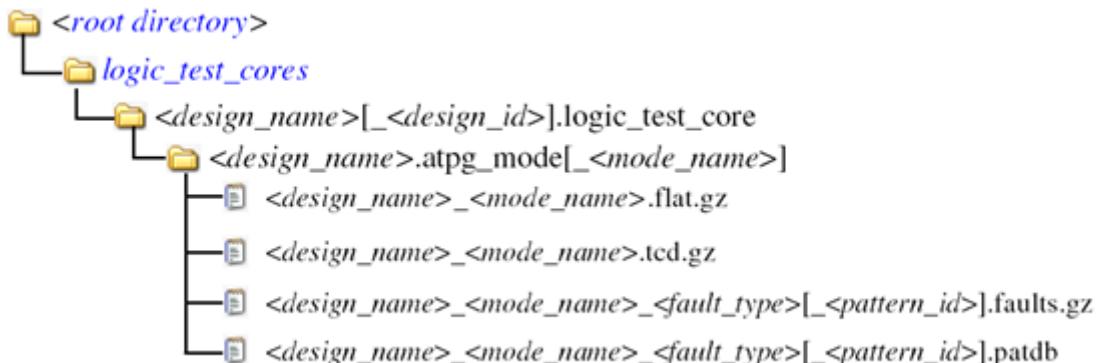
- <design_name>_<mode_name>_<fault_type>[_<pattern_id>].faults.gz

A file containing the fault classification for a given fault type and a specific pattern id that will later be used to create a consolidated fault coverage report per fault model. The pattern_id is specified using the -pattern_id option when invoking the [write_tsdb_data](#) command.

<design_name>.atpg_mode[_<mode_name>]

A directory used to store the information related to a logic core that has been configured for ATPG, passed design rule checks, and for which ATPG patterns were created.

Summary



Description

A directory used to store the information related to a logic core that has been configured for ATPG, passed design rule checks, and for which ATPG patterns were created. One typical usage is to store the ATPG results after generating retargetable patterns for a wrapped core. See also the [write_tsdb_data](#) command.

Components

- <*design_name*>.atpg_mode[<*mode_name*>]

A directory used to store the information for a given ATPG test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the [set_context](#) command. The mode name is specified using the [set_current_mode](#) command.

- <*design_name*>_<*mode_name*>.flat.gz

A file containing the flat model representing the DRCed ATPG test mode. The primary purpose of this file is for diagnosis. This file is also used to fault simulate other fault models at a later time without having to redo DRC.

- <*design_name*>_<*mode_name*>.tcd.gz

A file containing a Core wrapper describing different aspect of the ATPG test mode. The TCD file for the ATPG mode is used for pattern retargeting. Similarly, for top-level ATPG, it can be used for the core mapping flow. See “[Scan Pattern Retargeting](#)” and “[Core Mapping for ATPG Process Overview](#)” in the *Tessent Scan and ATPG User's Manual*.

- <*design_name*>_<*mode_name*>_<*fault_type*>[_<*pattern_id*>].faults.gz

A file containing the fault classification for a given fault type and a specific pattern id that will later be used to create a consolidated fault coverage report per fault model. The *pattern_id* is specified using the -pattern_id option when invoking the [write_tsdb_data](#) command.

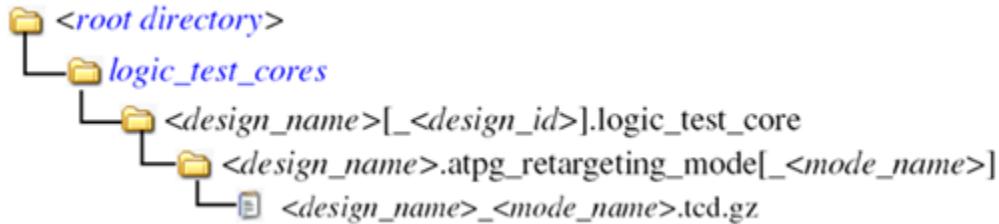
- <*design_name*>_<*mode_name*>_<*fault_type*>[_<*pattern_id*>].patdb

A file containing the ATPG pattern information.

<design_name>.atpg_retargeting_mode[<mode_name>]

A directory used to store the information related to a logic core that has been configured for retargeted ATPG, passed design rule checks, and for which retargeted ATPG patterns were created.

Summary



Description

A directory used to store the information related to a logic core that has been configured for retargeted ATPG, passed design rule checks, and for which retargeted ATPG patterns were created. See also the [write_tsdb_data](#) command and “Scan Pattern Retargeting” in the *Tessent Scan and ATPG User’s Manual*.

Components

- **<design_name>.atpg_retargeting_mode[_<mode_name>]**

A directory used to store the information for a given ATPG retargeting test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the [set_context](#) command. The mode name is specified using the [set_current_mode](#) command. The ATPG retargeting mode does not require that you have done pattern retargeting first. You need to have done at least done core extraction for pattern retargeting; see “[Core-in-Core Pattern Retargeting](#)” in the *Tessent Scan and ATPG User’s Manual* for more information.

- **<design_name>_<mode_name>.tcd**

A file containing a Core wrapper describing different aspect of the ATPG retargeting test mode. The TCD for the retargeting mode can be used for the following: pattern retargeting in a subsequent step to core extraction without requiring extraction or even a design; and/or for reverse mapping of tester failures for core-level diagnosis.

Chapter 13

Instrument Connectivity Language (ICL)

This chapter specifies the syntax of the Instrument Connectivity Language (ICL) of the IEEE 1687-2014 (IJTAG) standard as implemented by Tessent IJTAG.

For more information about Tessent IJTAG, refer to the *Tessent IJTAG User's Manual*.

ICL Overview	3783
Unsupported ICL Language Elements	3783
Common Features of the ICL Language.....	3784
ICL Statement Descriptions	3788
Syntax Conventions	3789
Module	3790
Attribute	3792
ScanInterface	3794
Chain.....	3800
ScanInPort.....	3803
ShiftEnPort	3805
CaptureEnPort.....	3807
UpdateEnPort	3809
DataInPort.....	3811
SelectPort	3814
ResetPort.....	3816
TMSPort	3818
TCKPort	3821
ClockPort	3823
TRSTPort	3825
AddressPort.....	3826
WriteEnPort	3828
ReadEnPort	3830
ScanOutPort	3832
DataOutPort	3835
ToShiftEnPort	3838
ToCaptureEnPort	3842
ToUpdateEnPort	3846
ToResetPort	3850
ToSelectPort	3853
ToClockPort	3856
ToTCKPort	3861
ToTMSPort	3863
ToTRSTPort	3865

ToIRSelectPort	3867
Instance	3869
Parameter	3875
LocalParameter	3878
Enum	3879
Alias	3882
LogicSignal	3885
ScanRegister	3887
DataRegister	3892
ScanMux	3897
DataMux	3899
ClockMux	3901
OneHotDataGroup	3903
OneHotScanGroup	3907
AccessLink	3909
NameSpace	3918
UseNameSpace	3919

ICL Overview

The syntax described in this chapter is not based on the formal grammar used in the IJTAG standard, but rather through a hierarchical description of an ICL Module and its elements.

Examples and suggestions for usage are provided for each language element. These elements are divided into language statements; that is, all the elements in the direct body of the ICL **Module** definition (plus a few more), and parameters. Parameters to an ICL language statement further define the properties of the statement. For example, the ICL language statement **ScanRegister** has the parameter **ResetValue**, which defines which value the scan register will assume if the reset signal turns active.

Unsupported ICL Language Elements	3783
Common Features of the ICL Language.....	3784
Numbers	3784
Strings	3785
Identifiers	3785
Limited Identifier Mathematics	3786
Concatenations	3786

Unsupported ICL Language Elements

Most of the language elements of the IEEE 1687 standard are supported. Currently unsupported ICL language elements or language elements with limited support are clearly marked.

For example:

Note
 The DefaultLoadValue parameter is currently not supported within the ScanInterface.

Common Features of the ICL Language

The following sections introduce common features of the language.

Numbers	3784
Strings	3785
Identifiers	3785
Limited Identifier Mathematics	3786
Concatenations	3786

Numbers

ICL recognizes unsigned integer numbers in any of the following three number base formats: decimal, hexadecimal, and binary.

Unsized decimal numbers may be composed of a sequence of the digits 0 to 9. If needed, their size will be determined based on context.

Sized numbers are composed of a size constant, followed by an optional white space and a base token (d or D for decimal, h or H for hexadecimal, and b or B for binary), and the digits of the number, expressed in the respective number base. The size constant always expresses the number of binary digits of the sized number, independent of the used base format. For example, the sized numbers 4'b1011, 4'hb and 4'd11 are identical and exactly 4 binary digits wide.

Binary, decimal, and hexadecimal numbers may use their respective common digit symbols (0 and 1 for binary, 0 to 9 for decimal, 0 to 9 and a to f for hexadecimal). These digit symbols are case insensitive. In addition, binary and hexadecimal numbers may use x or X to denote the unknown digit value.

Padding is applied automatically as necessary if the number of digits of the sized number is less than the size constant requires. For example, assume this comparison:

```
LogicSignal test { dataIn[7:0] == 8'b10 ; }
```

The “==” operator compares two 8-bit (that is, 8 binary digit wide sized numbers). However, only 2 digits “10” are given. Padding is executed based on the following rules: If the left most digit is non-x, then the padding adds as many 0 to the left as necessary. If however the leftmost digit is x, then padding adds as many x to the left as necessary. The above example will therefore be padded to the following:

```
LogicSignal test { dataIn[7:0] == 8'b00000010 ; }
```

The opposite of padding is truncation. Truncation happens if the number of digits of the sized number is larger than the size constant requires. Consequently, excessive digits are removed

from the left until the size requirement is matched. However, only occurrences of 0 or x may be removed. Removing any other digit symbol results in an error.

Strings

Escaping in ICL strings is not compliant with the IEEE 1687-2014 standard. This has been done to maintain backward compatibility with existing ICL files generated by Mentor Graphics tools.

Backslashes in ICL strings are interpreted as escaping indicators only if the next character is a backslash or double quotes. If the next character is something else, the backslash is interpreted as an ordinary character of the string.

Identifiers

IJTAG differentiates between four identifiers: The scalar identifier, the parameter identifier, the vector identifier, and the pin identifier.

The scalar identifier is used to denote the elements of the ICL netlist such as the ports of a module, a module name or an instance name. The scalar identifier is case sensitive and must start with a letter. Thereafter, the scalar identifier may use letters, numbers, and the underscore symbol. It may not contain any white spaces nor the "[" or "]" symbols (these are used for vector identifiers).

The parameter identifier is used to reference a [Parameter](#) or [LocalParameter](#) definition. The parameter reference is composed of the "\$" symbol and the scalar identifier name of the referenced (local) parameter. For example, if the parameter scalar identifier is `userValue`, the parameter reference will be `$userValue`.

There are two types of vector identifiers: The single index and the range vector identifier. There is no length limit for these identifiers.

A single index vector identifier is composed of a scalar identifier, followed by the "[" symbol, an unsigned decimal number, followed by the "]" symbol. Its width is considered to be 1. An example is `bus[4]`, denoting the 4th bit of the scalar identifier `bus`.

A ranged vector identifier is composed of a scalar identifier, followed by the "[" symbol, a first unsigned decimal number, the ":" symbol, a second unsigned decimal number, and followed by the "]" symbol. An example is:

```
bus [0 : 4]
```

denoting 5 bits of the scalar identifier `bus`. Note that the range can either be increasing, for example `[0:4]`, or decreasing, for example `[4:0]`.

Pin identifiers refer to a port name, optionally including an instance path name. The port name is either a scalar, a single index or range index vector identifier. The instance names are all

scalar identifiers, and the instance path is composed of these scalar identifiers, separated by the “.” symbol. An example is:

```
top.myInstA.myInstB.dataIn[7:0]
```

In this example the identifiers “top”, “myInstA” and “myInstB” are all denoting names of instances, whereas dataIn[7:0] is a ranged vector identifier of a port scalar named dataIn.

Limited Identifier Mathematics

Parameter identifiers, single indexes and ranged indexes may use limited mathematical operations: + (addition), - (subtraction), * (multiplication), / (division), and % (modulo). For parameter identifiers, a parameter value can be computed using these operations; for vector identifiers any used unsigned decimal number can be expressed using these operations. In addition, parameter reference may be used.

Examples are:

```
Parameter ten = 10 ;
Parameter twenty = 2*$ten ;
bus[$twenty:$ten]
bus[$twenty%2]
bus[$ten-1:0]
```

Concatenations

Under certain conditions, identifiers might be concatenated with other identifiers and/or sized and unsized numbers. The concatenation operator is the “,” symbol.

A first example is the following:

```
Parameter topBit 1'b1 ;
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, 0, 4'b1111 ; }
```

Observe that the left hand side of the comparator (==) is 8 bits wide. The right hand side has 2 sized numbers of a total width of 5 bits. Therefore the unsized number 0 in the middle will be auto-expanded into 3'b000. It is obvious that such a comparison construct may contain at most one unsigned number. This example shows also the usage of parameter references in lieu of a number.

A concatenation may also include the “~” symbol, denoting a sized, bit-wise, binary inversion. For example: ~4' ha is equivalent to 4'h5.

Interestingly, the “~”symbol can also be used within the concatenation:

```
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, ~0, 4'b1111 ; }
```

Here, the unsized, but inverted 0 is auto-expanded into 4'b1111.

ICL Statement Descriptions

The following describes each ICL statement available in Tessent IJTAG.

Syntax Conventions	3789
Module	3790
Attribute	3792
ScanInterface	3794
Chain	3800
ScanInPort	3803
ShiftEnPort	3805
CaptureEnPort	3807
UpdateEnPort	3809
DataInPort	3811
SelectPort	3814
ResetPort	3816
TMSPort	3818
TCKPort	3821
ClockPort	3823
TRSTPort	3825
AddressPort	3826
WriteEnPort	3828
ReadEnPort	3830
ScanOutPort	3832
DataOutPort	3835
ToShiftEnPort	3838
ToCaptureEnPort	3842
ToUpdateEnPort	3846
ToResetPort	3850
ToSelectPort	3853
ToClockPort	3856
ToTCKPort	3861
ToTMSPort	3863
ToTRSTPort	3865
ToIRSelectPort	3867
Instance	3869

Parameter	3875
LocalParameter	3878
Enum	3879
Alias	3882
LogicSignal	3885
ScanRegister	3887
DataRegister	3892
ScanMux	3897
DataMux	3899
ClockMux	3901
OneHotDataGroup	3903
OneHotScanGroup	3907
AccessLink	3909
NameSpace	3918
UseNameSpace	3919

Syntax Conventions

This chapter uses these syntax conventions when documenting ICL syntax properties.

Table 13-1. SyntaxConventions for ICL Files

Convention	Example	Usage
<i>Upright</i>	<code>ScanInPort port_name ;</code>	An upright font indicates an ICL language element, either a statement or a parameter.
<i>Italic</i>	<code>scan_in : port_pin_name;</code>	An italic font indicates a user-supplied value.
<u>Underline</u>	<code>wgl_type : generic lsi;</code>	An underlined item indicates the default value.
	<code>logic_level : both high low;</code>	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	<code>port_naming : port_naming, ...;</code>	Ellipses indicate a repeatable value. The comment “// repeatable” also indicates a repeatable value.
//	<code>// default: ijtag_so</code>	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

Module

Defines the name of an ICL module.

Usage

```
Module module_name {
    Attribute att_name = att_value;
    Parameter param_name = param_value;
    LocalParameter param_name = param_value;
    ScanInterface interface_name { ... }

    //---- Port Functions
    ScanInPort port_name { ... }
    ShiftEnPort port_name { ... }
    CaptureEnPort port_name { ... }
    UpdateEnPort port_name { ... }
    DataInPort port_name { ... }
    DataOutPort port_name { ... }
    SelectPort port_name { ... }
    ResetPort port_name { ... }
    TMSPort port_name { ... }
    TCKPort port_name { ... }
    ClockPort port_name { ... }
    TRSTPort port_name { ... }
    AddressPort port_name { ... }
    WriteEnPort port_name { ... }
    ReadEnPort port_name { ... }
    ScanOutPort port_name { ... }
    ToShiftEnPort port_name { ... }
    ToUpdateEnPort port_name { ... }
    ToCaptureEnPort port_name { ... }
    ToResetPort port_name { ... }
    ToSelectPort port_name { ... }
    ToClockPort port_name { ... }
    ToTCKPort port_name { ... }
    ToTMSPort port_name { ... }
    ToTRSTPort port_name { ... }
    ToIRSelectPort port_name { ... }

    Instance instance_name Of another_module_name { ... }
    LogicSignal signal_name { ... }
    ScanRegister register_name { ... }
    DataRegister register_name { ... }
    ScanMux mux_name SelectedBy selector { ... }
    DataMux mux_name SelectedBy selector { ... }
    ClockMux mux_name SelectedBy selector { ... }
    OneHotDataGroup name { ... }
    OneHotScanGroup name { ... }
    Enum enum_name { ... }
    Alias alias_name = element_list { ... }
    AccessLink instance_name Of link_type { ... }
}
```

Description

The syntax above lists the complete contents (statements) of an ICL module. Not all statements may be used at the same time.

Each ICL statement is explained in detail in the following pages. All statements are hyperlinked from here to the respective statement description, as well as back to this module description.

Arguments

- `module_name`

The `Module` statement defines the name of an ICL module. The name of the module must be unique among all loaded modules. To build an ICL design, you use the [Instance](#) statement.

Related Topics

[NameSpace](#)

[Instance](#)

[UseNameSpace](#)

Attribute

Specifies a user attribute name and value.

Usage

```
Module module_name {  
    Attribute att_name = att_value ;  
}
```

Description

The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool.

Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual.

Note that most Tessent attributes are read-only and are automatically generated by the Tessent tool, which also assigns a value to the attribute.

The Attribute statement is not only allowed in [Module](#), but in most ICL objects documented in this chapter.

Note

 In its usage within ICL objects, like modules, ports, scan or data mux, and so on, the Attribute statement is always optional.

Arguments

- *att_name = att_value*

An required string and optional “=”symbol and value combination, where *att_name* defines the name of the Attribute, and *att_value*, if present, defines the value of the attribute.

Within an ICL object where multiple Attribute statements are used, each *att_name* must be unique.

Examples

Example 1

The first example shows a very simple usage of assigning a string “true” to an attribute name “is_top”:

```
Module top {  
    Attribute is_top = "true" ;  
    [...]  
}
```

Example 2

The second example shows a Tessent attribute connection_rule_option that influences ICL Network Extraction. Please see [Performing ICL Extraction](#) in the *Tessent IJTAG User's Manual* for the details of the attribute.

```
DataInPort din {Attribute connection_rule_option = "allowed_tied_high";}
```

ScanInterface

Defines a set of ports of the module that together perform a scan operation.

Usage

```
Module module_name {
    ScanInterface interface_name {
        Attribute att_name = att_value ;
        Port port_name ;
        DefaultLoadValue load_value ;
        Chain chain_name { ... }
    }
}
```

Description

Defines a set of ports of the module that together perform a scan operation.

The ScanInterface defines a set of ports of the module that together perform a scan operation. A typical example of ports include the shift enable, capture, and update enable ports, together with at least a pair of scan-in and scan-out ports. However, the ScanInterface is applied also to the TAP, since a shift-based interface to or from a module is also defined; that is, a scan interface. See the discussion on *port_names* below, which defines the sets of ports of the four recognized use models of the ScanInterface — the client interface, the host interface, the client TAP interface, and the host TAP interface. A module may have more than one scan interface, in particular several client interfaces or interfaces for different use models. However, only one scan interface may be active at the same time. This means, for example, that a set of scan chains that are supposed to be operated together must all be in the same ScanInterface definition.

For a handoff module, with more than one portfunction of type [ScanInPort](#) or [ScanOutPort](#), there must be as many ScanInterface statements as there are scan interfaces in the module. This does not mean, for example, that the number of scan-in and scan-out port pairs defines the number of ScanInterface statements. Since several pairs of scan-in and scan-out ports may belong to the same scan interface, which means they are supposed to be operated simultaneously, the number of ScanInterface statements may be smaller.

The examples below are for simple ScanInterface definitions. Additional examples, especially those for multiple parallel scan chains, are provided in the ScanInterface/[Chain](#) section.

Arguments

- *interface_name*
A required string that defines the name of the scan interface of the module.
- Attribute *att_name = att_value* ;
A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool.

- Port *port_name* ;

A repeatable, optional keyword-string pair that references the name of an existing port of the module. Four use models of a ScanInterface can be distinguished. The port_names for a given ScanInterface must all belong to the same use model.

- The client interface — Through a client scan interface a module receives all signals necessary to operate a defined set of scan chains. Ports of the following port functions form a client scan interface: at least one pair of [ScanInPort](#) and [ScanOutPort](#), one [ShiftEnPort](#) and/or at least one [SelectPort](#), and [CaptureEnPort](#), [UpdateEnPort](#), [ResetPort](#), or [TCKPort](#). There can be more than one CaptureEnPort or more than one UpdateEnPort in one client interface, as long as all ports with the same port function have distinct function_modifier attributes. If the shift enable, capture enable, or update enable ports are not stated and the module has exactly one of each, these ports are implied for the scan interface using the modules set of port functions, or for non-handoff modules, the set or implied set of port functions. If more than one pair of scan-in and scan-out ports belong to the same scan interface, the [Chain](#) object must be used. In this case, no port name should reference a scan-in and scan-out port, nor should the DefaultLoadValue be used.

If a non-handoff module has exactly one client interface, than the ScanInterface statement is not required. The tool assumes a “default” client interface.

- The host interface — Through a host scan interface a module defines shift operating control signals for another module. Ports of the following port functions form a host scan interface: one [ScanInPort](#) and/or one [ScanOutPort](#), one [ToShiftEnPort](#) and/or at least one [ToSelectPort](#), and [ToCaptureEnPort](#), [ToUpdateEnPort](#), [ToResetPort](#), or [ToTCKPort](#). There can be more than one ToCaptureEnPort or more than one ToUpdateEnPort in one host interface, as long as all ports with the same port function have distinct function_modifier attributes. Similar to the client interface, if the module has exactly one [ToShiftEnPort](#), [ToCaptureEnPort](#), or [ToUpdateEnPort](#), these ports are assumed for each host scan interface, for which they have not been stated already.
- The client TAP interface — Through a client TAP interface, the module receives all signals necessary to operate a TAP controller: One of each [ScanInPort](#), [ScanOutPort](#), and [TMSPort](#), and at most one of [TRSTPort](#) and [TCKPort](#).
- The host TAP interface — Through a host TAP interface, the module defines signals for another module’s client TAP interface: One [ScanInPort](#) and/or one [ScanOutPort](#), one [ToTMSPort](#), and at least one of [ToTRSTPort](#) and [ToTCKPort](#).

Note

 ScanInterfaces with multiple CaptureEnPorts, UpdateEnPorts, ToCaptureEnPorts or ToUpdateEnPorts are not standard-compliant. Third-party IJTAG tools might reject the processing of ICL files with such ScanInterface specifications.

- DefaultLoadValue *load_value* ;
An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. The DefaultLoadValue property affects the behavior of the iScan command only. The iScan command is currently not supported.
- Chain *chain_name* ;
A repeatable, optional keyword-string pair that declares the name of a scan chain of the module. All chain names must be unique within a ScanInterface. If the ScanInterface has only one [ScanInPort/ScanOutPort](#) pair, than [Chain](#) object is not required.

Examples

Example 1

The following example shows three equivalent description of an ICL module with a single scan-in / scan-out client scan interface. This is not a complete list, other variations are possible. Note that for the module tdr4_1, no ScanInterface statement is given. It is not necessary, since the module has exactly one client scan interface. However, you may provide a ScanInterface as shown for modules tdr4_2, and tdr4_3.

```

Module tdr4_1 {
    ScanInPort      si;
    ScanOutPort     so      { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource  si;
        ResetValue    8'b0 ;
    }
}
Module tdr4_2 {
    ScanInPort      si;
    ScanOutPort     so      { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanInterface interface1 {
        Port en ;
        Port si; Port so;
    }
    ScanRegister R[7:0] {
        ScanInSource  si;
        ResetValue    8'b0 ;
    }
}
Module tdr4_3 {
    ScanInPort      si;
    ScanOutPort     so      { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanInterface interface2 {
        Port se ;
        Chain chain1 {
            Port si; Port so;
        }
    }
    ScanRegister R[7:0] {
        ScanInSource  si;
        ResetValue    8'b0 ;
    }
}

```

Example 2

The next example shows the port functions and both the client and host TAP interfaces of a typical TAP controller module.

```

Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux;  }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn     {  Source sel;   }
    ScanInPort   fromTdr;
    ToSelectPort tapEn     {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;
    ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }
    ScanInterface hostTAP  {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }
    [...]
}

```

Example 3

The next example shows an example implementation of a SIB (Segment Insertion Bit) module. This module implements a scan chain switch by selecting either of two scan input ports. The module also generates an outgoing enable signal that is correlated to the second scan-in port.

```

Module sib {
    ScanInPort      si;
    ScanOutPort     so  {  Source SIB;  }
    ShiftEnPort    se;
    CaptureEnPort  ce;
    UpdateEnPort   ue;
    SelectPort     en;
    TCKPort        tck;
    ScanInPort     fso;
    ToSelectPort   ten {  Source And;  }
    ScanInterface client {
        Port si; Port so;
        Port se; Port ce; Port ue; Port en;
    }
    ScanInterface host  {
        Port fso; Port ten;
    }
    ScanRegister SIB {
        ResetValue    1'b0;
        ScanInSource Mux;
    }
    ScanMux Mux SelectedBy SIB {
        1'b0 : si;
        1'b1 : fso;
    }
    LogicSignal And { en,SIB == 2'b11;  }
}

```

Related Topics

[Chain](#)
[ScanInPort](#)
[SelectPort](#)
[ShiftEnPort](#)
[CaptureEnPort](#)
[UpdateEnPort](#)
[ResetPort](#)
[TCKPort](#)
[TMSPort](#)
[TRSTPort](#)
[ScanOutPort](#)
[ToShiftEnPort](#)
[ToCaptureEnPort](#)
[ToUpdateEnPort](#)
[ToResetPort](#)
[ToTCKPort](#)
[ToTMSPort](#)
[ToTRSTPort](#)
[ScanRegister](#)
[ScanMux](#)
[LogicSignal](#)

Chain

Defines the name of a scan chain within a ScanInterface.

Usage

```
ScanInterface interface_name {
    Chain chain_name {
        Attribute att_name = att_value ;
        Port port_name ;
        DefaultLoadValue load_value ;
    }
}
```

Description

The Chain statement defines the name of a scan chain within a [ScanInterface](#).

This statement does not create a scan register in the sense of the [ScanRegister](#) statement. Here it is only a declaration which [ScanInPort](#) and [ScanOutPort](#) of the module will be operated together with the control signals of the [ScanInterface](#). There can be more than one Chain statement in a [ScanInterface](#). In this case, each scan chain name must be unique.

Arguments

- Chain *chain_name* ;
A repeatable, required keyword-string pair that defines the name of a scan chain within a given ScanInterface. There can be more than one scan chain in any given ScanInterface. However, the names of the scan chains within a ScanInterface must all be unique.
- Attribute *att_name = att_value* ;
A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent does not have any built-in attributes for the ScanInterface/Chain statement.
- Port *port_name* ;
A required keyword-string pair, with the *port_name* referencing a [ScanInPort](#) or [ScanOutPort](#) port of the module. Within each Chain statement, there are exactly two Port statements, one referencing the scan-in port, and the other referencing the scan-out port.
- DefaultLoadValue *load_value* ;
An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. The DefaultLoadValue property affects the behavior of the iScan command only. The iScan command is currently not supported.

Examples

Example 1

The following example shows a module with two client ScanInterfaces, each operating two scan chains.

```
Module chip1 {
    ScanInPort siA1 ;
    ScanInPort siA2 ;
    ScanOutPort soA1 { Source blockA.so1 ; }
    ScanOutPort soA2 { Source blockA.so2 ; }
    ScanInPort siB1 ;
    ScanInPort siB2 ;
    ScanOutPort soB1 { Source blockB.so1 ; }
    ScanOutPort soB2 { Source blockB.so2 ; }
    ShiftEnPort sel1 ;
    CaptureEnPort cel1 ;
    UpdateEnPort uel ;     ShiftEnPort se2 ;
    CaptureEnPort ce2 ;
    UpdateEnPort ue2 ;     TCKPort tck ;
    ScanInterface cA12 {
        Port sel1; Port cel1; Port uel;
        Chain c1 { Port siA1; Port soA1 ; }
        Chain c2 { Port siA2; Port soA2 ; }
    }
    ScanInterface cB12 {
        Port se2; Port ce2; Port ue2;
        Chain c1 { Port siB1; Port soB1 ; }
        Chain c2 { Port siB2; Port soB2 ; }
    }
}
[...]
```

Example 2

In this variation of Example 1 above, the [ShiftEnPort](#), [CaptureEnPort](#), and [UpdateEnPort](#) ports are shared among both [ScanInterface](#). The selection between the two interfaces is accomplished through the two [SelectPort](#) sel1 and sel2, respectively.

```
Module chip2 {
    ScanInPort siA1 ;
    ScanInPort siA2 ;
    ScanOutPort soA1 { Source blockA.so1 ; }
    ScanOutPort soA2 { Source blockA.so2 ; }
    ScanInPort siB1 ;
    ScanInPort siB2 ;
    ScanOutPort soB1 { Source blockB.so1 ; }
    ScanOutPort soB2 { Source blockB.so2 ; }
    SelectPort selA ;
    SelectPort selB ;    ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;    TCKPort tck ;    ScanInterface cA12 {
        Port selA;
        Chain c1 { Port siA1; Port soA1 ; }
        Chain c2 { Port siA2; Port soA2 ; }
    }           ScanInterface cB12 {
        Port selB;
        Chain c1 { Port siB1; Port soB1 ; }
        Chain c2 { Port siB2; Port soB2 ; }
    }
    [...]
}
```

Example 3

This example is another variation of Example 1. The difference is that all ports are busses and not individual ports as in Example 1. The operation of the modules is identical.

```
Module chip3 {
    ScanInPort si[3:0] ;
    ScanOutPort so[3:0] { Source blockA.so1, blockA.so2,
                         blockB.so1, blockB.so2 ;
                         }    ShiftEnPort se[1:0] ;
    CaptureEnPort ce[1:0] ;
    UpdateEnPort ue[1:0] ;    TCKPort tck ;    ScanInterface cA12 {
        Port se[0]; Port ce[0]; Port ue[0];
        Chain c1 { Port si[0]; Port so[0] ; }
        Chain c2 { Port si[1]; Port so[1] ; }
    }           ScanInterface cB12 {
        Port se[1] ; Port ce[1]; Port ue[1];
        Chain c1 { Port si[2]; Port so[2] ; }
        Chain c2 { Port si[3]; Port so[3] ; }
    }
    [...]
}
```

Related Topics

[ScanInterface](#)

[ScanRegister](#)

ScanInPort

Declares the name of a scan input port of a Module.

Usage

```
Module module_name {
    ScanInPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of a scan input port of a [Module](#).

A module may have zero or more scan-in ports. However, if a module has at least one [ScanOutPort](#) defined, then there must be at least one ScanInPort.

If a module has exactly one ScanInPort, than the module may also have a [ScanInterface](#) defined. However, if a module has more than one ScanInPort, than there must be at least one [ScanInterface](#) definition.

The scan-in port is used to execute a serial write (that is, not a parallel write operation). If a parallel write is to be done, use the [DataInPort](#) port function instead.

Arguments

- *port_name*

A required string that identifies the name of a scan-in port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name = att_value ;*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no [ScanInterface](#) is required. The scan-in port is named “si”. Observe that the port name is reused in the body of the [ScanRegister](#) R[7:0] declaration. There, it declares that the

scan register will receive its scan data from the object named “si”, which happens to be the ScanInPort of the Module.

```
Module tdr2 {
    ScanInPort      si;
    ScanOutPort     so { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource   si;
        ResetValue     8'b0 ;
        DefaultLoadValue 8'b0 ;
    }
}
```

Related Topics

[CaptureEnPort](#)
[UpdateEnPort](#)
[ShiftEnPort](#)
[ScanOutPort](#)
[ScanInterface](#)
[DataInPort](#)

ShiftEnPort

Declares the name of the shift enable port of a Module's client ScanInterface.

Usage

```
Module module_name {
    ShiftEnPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the shift enable port of a [Module](#)'s client [ScanInterface](#).

This port is used to enable the shift operation into a module's [ScanRegister](#) elements. A module may have zero or more shift enable ports. For an internal module; that is, not a hand-off module, a single shift enable port is implied if there are scan in/out ports in the module. In this case, a ShiftEnPort need not be declared. The ShiftEnPort must be declared if:

- The module is a hand-off module.
- There is more than one shift enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The shift enable signal is an active high signal. A ShiftEnPort of an instance is sourced by a ToShiftEnPort. A [ScanRegister](#) object is expected to hold its scan data, as long as the shift enable port is low, independently on the number of clock cycles.

Arguments

- *port_name*

A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ShiftEnPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name = att_value ;*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin_attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client [ScanInterface](#) is required. The shift enable port is named “se”. Observe that the body of the Module does not further use the shift enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the [ScanInterface](#) to learn how you can use multiple scan chains, potentially with multiple shift enable signals.

```
Module tdr2 {
    ScanInPort      si;
    ScanOutPort     so { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource   si;
        ResetValue     8'b0 ;
        DefaultLoadValue 8'b0 ;
    }
}
```

Related Topics

[CaptureEnPort](#)
[UpdateEnPort](#)
[ScanInPort](#)
[ScanOutPort](#)
[ScanInterface](#)

CaptureEnPort

Declares the name of the capture enable port of a Module's client ScanInterface.

Usage

```
Module module_name {  
    CaptureEnPort port_name {  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the capture enable port of a [Module](#)'s client [ScanInterface](#).

This port is used to control the capture operation of a module's [ScanRegister](#) elements. A module may have zero or more capture enable ports. For an internal module; that is, not a hand-off module, a single capture enable port is implied if there are scan in/out ports in the module. In this case, a CaptureEnPort need not be declared. The CaputureEnPort must be declared if:

- The module is a hand-off module.
- There is more than one capture enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The capture enable signal is an active high signal. A CaptureEnPort of an instance is sourced by a CaptureEnPort of the parent module, a [ToCaptureEnPort](#) of another instance or a DataMux which has a capture signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the capture enable pulse is assumed to be suppressed.

Note

 The suppressing of the enable signal described above is currently not supported in Tessent.

Arguments

- *port_name*

A required string that identifies the name of a capture enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL CaptureEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the

language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client [ScanInterface](#) is required. The capture enable port is named “ce”. Observe that the body of the Module does not further use the capture enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the [ScanInterface](#) to learn how you can use multiple scan chains, potentially with multiple capture enable signals.

```
Module tdr2 {
    ScanInPort      si;
    ScanOutPort     so { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource   si;
        ResetValue     8'b0 ;
        DefaultLoadValue 8'b0 ;
    }
}
```

Related Topics

[ShiftEnPort](#)
[UpdateEnPort](#)
[ScanInPort](#)
[ScanOutPort](#)
[ScanInterface](#)

UpdateEnPort

Declares the name of the update enable port of a Module's client ScanInterface.

Usage

```
Module module_name {
    UpdateEnPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the update enable port of a [Module](#)'s client [ScanInterface](#).

This port is used to control the update operation of a module's [ScanRegister](#) elements. A module may have zero or more capture enable ports. A module may have zero or more update enable ports. For an internal module; that is, not a hand-off module, a single update enable port is implied if there are scan in/out ports in the module. In this case, an UpdateEnPort need not be declared. The UpdateEnPort must be declared if:

- The module is a hand-off module.
- There is more than one update enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The update enable signal is an active high signal. An UpdateEnPort of an instance is sourced by an UpdateEnPort of the parent module, a ToUpdateEnPort of another instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed. An UpdateEnPort of an instance is sourced by an UpdateEnPort of the parent module, a ToUpdateEnPort of another instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed.

Note

 The suppressing of the enable signal described above is currently not supported in Tessent.

Arguments

- *port_name*

A required string that identifies the name of an update enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL UpdateEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client [ScanInterface](#) is required. The update enable port is named “ue”. Observe that the body of the Module does not further use the update enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the [ScanInterface](#) to learn how you can use multiple scan chains, potentially with multiple update enable signals.

```
Module tdr2 {
    ScanInPort      si;
    ScanOutPort     so { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource  si;
        ResetValue    8'b0 ;
        DefaultLoadValue 8'b0 ;
    }
}
```

Related Topics

[ShiftEnPort](#)
[CaptureEnPort](#)
[ScanInPort](#)
[ScanOutPort](#)
[ScanInterface](#)

DataInPort

Declares the name of the data in port of a Module.

Usage

```
Module module_name {
    DataInPort port_name {
        RefEnum enum_name ;
        DefaultLoadValue default_load_value ;
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the data in port of a [Module](#).

A module may have zero or more data in ports. The number and bus-width of all declared data in ports is not correlated to the number and bus-width of any [DataOutPort](#).

A data in port is used to execute a parallel write (that is, not a scan operation), to ports of the module. For a scan operation, use the [ScanInPort](#) port function instead. If parallel writes to ports of the module are allowed, the DataInPort function(s) must be declared for the module. If a module does not have DataInPort port functions, no parallel data write is allowed. Certain additional restrictions apply as explained next.

There should be at most one RefEnum per DataInPort, and the widths of the data within the referenced enumeration table should match the width of the data in port.

If a data in port is used to select a scan path, for example, as the select of a [ScanMux](#), it may not be target of a write operation. This ensures that the scan selection remains under control of the software application.

If a data in port is connected to an [Instance](#) or [DataRegister](#) with an AddressValue statement, all data in ports are concatenated in the order of declaration until the required data width is achieved. See the discussion of AddressValue for further explanation and an example.

Arguments

- *port_name*

A required string that identifies the name of a data in port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL DataInPort of the module that is also used for the respective ports of the module in the design.

- RefEnum *enum_name* ;

An optional keyword-string pair that references the name of an Enum table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. It must be understood that the enumeration table is not necessarily a complete list. It is still allowed to write to the data in ports any value, including those not listed in the enumeration table. A DataInPort may have at most one RefEnum statement.

- DefaultLoadValue *default_load_value* ;

An optional keyword-string pair that identifies the value that each DataIn port will assume if no user-defined value had been written to it beforehand and other bits of the Ports needs to be written to. If used, the width of *default_load_value* must match the width of the port.

For DataInPorts on the current design, the tool will automatically force those DataInPorts to the specified *default_load_value* at the beginning of the pattern. There is one exception to this behavior. If the DataInPort is declared as an inout port in the design module having the same name as the top level ICL module, the port is driven to Z at the beginning of the pattern and will remain Z until the port is the target of an explicit iWrite command or its value is needed to justify other iRead or iWrite commands.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessonst understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

Example 1

The following example shows an ICL module definition with only DataInPort, [DataOutPort](#), and a [ClockPort](#) port functions. This module does not include any scan interface. Its only way of data IO is through the parallel data port in[7:0] and out[7:0]. Observe that the body of the module is empty, not containing any other ICL objects, like [DataRegister](#), or [LogicSignal](#).

```
Module instrument {
    DataInPort  in[7:0];
    DataOutPort out[7:0];
    ClockPort   clk;
}
```

Example 2

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how the data in port is used inside the [ScanRegister](#). It is declared as the source of the capture values of the scan register, which means that the scan register will capture the value from the data in ports fq on the positive edge of the [CaptureEnPort](#) ce, under the condition that the [SelectPort](#) en is also high. Finally, observe the usage of the RefEnum and [Enum](#) statements.

```

Module tdr_ref {
    ScanInPort      si;
    ScanOutPort     so      {  Source R[0];   }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    DataInPort      fq[7:0]  {  RefEnum OptionalValues;  }
    DataOutPort     td[7:0]  {  Source R[7:0]; RefEnum OptionalValues;  }
    ScanRegister R[7:0] {
        CaptureSource fq[7:0];
        ScanInSource  si;
        RefEnum       OptionalValues ;
    }   Enum OptionalValues {
        reset = 8'b0 ;
        set   = 8'b11111111 ;
        chain = 8'b11001100 ;
        green = 8'hee ;
        blue  = 8'd198 ;
    }
}

```

Example 3

The following example shows how a data input port can be used to select a scan path. In such a case, the data input port falls under the control of the Tessent, which will determine the necessary data in values to fulfill any read or write requests to the scan chains.

```

Module msel {
    ScanOutPort so {  Source smux;  }
    DataInPort  dataSel;
    ScanMux smux SelectedBy dataSel {
        1'b0 : tdr0[0];
        1'b1 : tdr1[0];
    }
    [...]
}

```

Related Topics

[DataOutPort](#)
[ScanInPort](#)
[ScanOutPort](#)
[Enum](#)

SelectPort

Declares the name of the select input port of a Module's client ScanInterface.

Usage

```
Module module_name {
    SelectPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the select input port of a [Module](#)'s client [ScanInterface](#).

A module may have zero or more select ports. It is used to enable a scan interface of a module. When the select port is active (high), the scan interface is active. However, if the select port is inactive (low), the scan interface will not respond to any other control signal defined for this scan interface. The SelectPort statement for a non-handoff module is optional and implied if the module has a scan interface. The SelectPort is only required if the port is part of the [ScanInterface](#) statement. A SelectPort which is not part of a scan interface and which is not part of the scan network configuration logic is equivalent to a DataInPort.

Arguments

- *port_name*

A required string that identifies the name of a select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL SelectPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

In this example, the [ShiftEnPort](#), [CaptureEnPort](#), and [UpdateEnPort](#) ports are shared among both [ScanInterface](#). The selection between the two interfaces is accomplished through the two [SelectPort](#) sel1 and sel2, respectively.

```
Module chip2 {
    ScanInPort siA1 ;
    ScanInPort siA2 ;
    ScanOutPort soA1 { Source blockA.so1 ; }
    ScanOutPort soA2 { Source blockA.so2 ; }
    ScanInPort siB1 ;
    ScanInPort siB2 ;
    ScanOutPort soB1 { Source blockB.so1 ; }
    ScanOutPort soB2 { Source blockB.so2 ; }
    SelectPort selA ;
    SelectPort selB ;
    ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;
    TCKPort tck ;
    ScanInterface cA12 {
        Port selA;
        Chain c1 { Port siA1; Port soA1 ; }
        Chain c2 { Port siA2; Port soA2 ; }
    }
    ScanInterface cB12 {
        Port selB;
        Chain c1 { Port siB1; Port soB1 ; }
        Chain c2 { Port siB2; Port soB2 ; }
    }
    [...]
}
```

Related Topics

[ScanInPort](#)

[DataInPort](#)

[ScanInterface](#)

[Chain](#)

ResetPort

Declares the name of the reset input port of a Module's client ScanInterface.

Usage

```
Module module_name {
    ResetPort port_name {
        Attribute att_name = att_value ;
        ActivePolarity 0 | 1 ;
    }
}
```

Description

A port function that declares the name of the reset input port of a [Module](#)'s client [ScanInterface](#).

You have the option to chose the active polarity of the port.

A ResetPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the ResetPort to become active under different conditions (synchronous reset, asynchronous reset, enforced local reset and so on). If the driver has not been specified explicitly, then the ResetPort may be implicitly connected to a reset signal in the parent module. See the chapter "[How to Model Global Reset, Local Reset and Embedded TAPs](#)" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ResetPort.

Arguments

- *port_name*

A required string that identifies the name of a reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ActivePolarity 0 | 1

An optional keyword-value pair, that defines if the reset port of the module is considered active low (value = 0), or active high (value =1). If the ActivePolarity statement is not given, the active polarity of the reset port is assumed to be high.

Related Topics

[ScanOutPort](#)

[ScanInterface](#)

TMSPort

Declares the name of the TMS input port of a Module's client TAP ScanInterface.

Usage

```
Module module_name {
    TMSPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the TMS input port of a [Module](#)'s client TAP ScanInterface.

If a module has a client TAP interface, the TMSPort declaration may not be omitted. If the module has a [ToIRSelectPort](#), the module must also have exactly one TMSPort.

IJTAG supports top-level TAPs, as well as any number of embedded TAP controllers.

A TMSPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the TMSPort either to be in its normal mode of operation or to be continuously driven with the value '1' (which causes the associated TAP controller to be held in "reset" state) or to be continuously driven with the value '0' (which causes the associated TAP controller to be held in "idle" state). If the driver has not been specified explicitly, then the TMSPort may be implicitly connected to a tms signal in the parent module. See chapter "[How to Model Global Reset, Local Reset and Embedded TAPs](#)" in the *Tessent IJTAG User's Manual* for the details on embedded TAPs and the different ways of connecting a TMSPort.

Arguments

- *port_name*

A required string that identifies the name of the TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#)

It is recommended to use a port name for the ICL TMSPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#)

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine.

```
Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux;  }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn1   {  Source sel1;   }
    ScanInPort   fromTdr1;
    ToSelectPort tapEn1   {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;
    ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }
    ScanInterface hostTAP  {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }
    Instance fsm_i Of fsm  {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }
    ScanRegister IR[2:0] {
        CaptureSource 3'b001;
        ResetValue     3'b000;
        ScanInSource   tdi;
    }

    ScanRegister bypass {
        CaptureSource 1'b0;
        ScanInSource   tdi;
    } ScanMux IRMux SelectedBy fsm_i.irSel {
        1'b0 : DRMux;
        1'b1 : IR[0];
    } ScanMux DRMux SelectedBy IR[2:0] {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    } LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
    TCKPort      tck;
    TMSPort      tms;
    TRSTPort     trst;
    ToIRSelectPort irSel;
    ToResetPort  tlr;
}
```

Related Topics

[ScanInterface](#)

[TRSTPort](#)

[ToIRSelectPort](#)

TCKPort

Declares the name of the test clock (TCK) input port of a Module.

Usage

```
Module module_name {
    TCKPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the test clock (TCK) input port of a [Module](#).

The declaration of the TCKPort is optional for an internal module (that is, not for a hand-off module), even if the module contains scan registers. In this case, the test clock is implicitly assumed.

A module may have more than one TCKPort statement. In this case, all declared test clocks will be considered equivalent and with the same polarity.

Non test clock ports are defined through the [ClockPort](#) statement.

Note that IJTAG has no means of defining the actual speed of a test clock. This is left to the software implementing the standard. In Tessent, you can use the [add_clocks](#) command to define clock properties, like the period or the off-value.

Arguments

- *port_name*

A required string that identifies the name of the test clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotScanGroup](#), [OneHotDataGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL TCKPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[ScanInterface](#)

[ScanRegister](#)

[ClockPort](#)

ClockPort

Declares the name of a functional clock input port of a Module.

Usage

```
Module module_name {
    ClockPort port_name {
        Attribute att_name = att_value ;
        DifferentialInvOf other_clock_port_name ;
    }
}
```

Description

A port function that declares the name of a functional clock input port of a [Module](#) (that is, a clock that is not a test clock).

A test clock is defined through the [TCKPort](#) statement. A module may have zero or more clock ports.

Note that IJTAG has no means of defining the actual speed of a functional clock. This is left to the software implementing the standard. In Tessent, you can use the [add_clocks](#) command to define clock properties, like the period, or if the clock is synchronous or asynchronous to the test clock. Inside of ICL, only the [ToClockPort](#) statement can be used to model an oscillator-like behavior.

Arguments

- *port_name*

A required string that identifies the name of a clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ClockPort of the module that is also used for the respective ports of the module in the design.

- DifferentialInvOf *other_clock_port_name* ;

An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example declares one test clock and two functional, non-test clocks, an ordinary one and a differential one, the latter of which consists of two clock pins “pdifff” and “ndiff”.

```
Module top {
    TCKPort tck;
    ClockPort sclk;
    ClockPort pdifff;
    ClockPort ndiff { DifferentialInvOf pdifff ; }
```

Related Topics

[TCKPort](#)

TRSTPort

Declares the name of a TAP reset input port of a Module's client TAP ScanInterface.

Usage

```
Module module_name {  
    TRSTPort port_name {  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of a TAP reset input port of a [Module](#)'s client TAP [ScanInterface](#). The active polarity of the port is low.

A TRSTPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the TRSTPort to become active under different conditions (asynchronous global reset, enforced local reset and so on). If the driver has not been specified explicitly, then the TRSTPort may be implicitly connected to a trst signal in the parent module. See chapter "[How to Model Global Reset, Local Reset and Embedded TAPs](#)" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a TRSTPort.

Arguments

- *port_name*

A required string that identifies the name of the TRST port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL TRSTPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[ResetPort](#)

[TMSPort](#)

[ScanInterface](#)

AddressPort

Declares the name of an address input port of a Module.

Usage

```
Module module_name {
    AddressPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of an address input port of a [Module](#). It is used if the module contains addressable instances or data registers.

This port serves as the address of the addressable instances and data registers.

Arguments

- *port_name*

A required string that identifies the name of an address port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL AddressPort of the module that is also used for the respective ports of the module in the design.

The source of the address port name signal may only be an address port of the parent module, or scan register bits of a [ScanRegister](#) statement. In both cases, this connection must be direct, and may not include any logic signals.

If a module has more than one AddressPort, the address comparison with logic signals of the module implicitly concatenates all address port signals in order of declaration (top to bottom).

- Attribute *att_name = att_value ;*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[WriteEnPort](#)

[Instance](#)

[ReadEnPort](#)

[DataRegister](#)

WriteEnPort

Declares the name of the write enable input port of a Module.

Usage

```
Module module_name {
    WriteEnPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the write enable input port of a [Module](#). It is used if the module contains addressable instances or data registers.

This port serves as the write enable to the addressable instance and data registers.

Arguments

- *port_name*

A required string that identifies the name of a write enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

The source of the write enable port name signal may only be a write enable port of the parent module, or scan register bits of a [ScanRegister](#) statement. In both cases, this connection must be direct, and may not include any logic signals.

A module may have at most one WriteEnPort statement. A module must have a WriteEnPort statement, if the module contains writable, addressable data registers or writable, addressable instances.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[Instance](#)

[ReadEnPort](#)

DataRegister

ReadEnPort

Declares the name of the read enable input port of a Module.

Usage

```
Module module_name {
    ReadEnPort port_name {
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the read enable input port of a [Module](#). It is used if the module contains addressable instances or data registers.

This port serves as the read enable to the addressable instance and data registers.

A read operation is executed in two steps. At first the ReadEnPort must be activated (high) and the address must be supplied ([AddressPort](#)). In the second step, the capture is executed and the data is being read. During this second step the ReadEnPort is automatically deactivated (low) by means of an automatic iApplyEndState property (see the [Alias](#) statement).

Arguments

- *port_name*

A required string that identifies the name of a read enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

The source of the read enable port name signal may only be a read enable port of the parent module, or scan register bits of a [ScanRegister](#) statement. In both cases, this connection must be direct, and may not include any logic signals.

A module may have at most one ReadEnPort statement. A module must have a ReadEnPort statement, if the module contains readable, addressable data registers or readable, addressable instances.

A read enable port is assumed to have an automatic iApplyEndState property. See the [Alias](#) statement to learn more about iApplyEndState.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#)

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[AddressPort](#)

[Alias](#)

[DataRegister](#)

[Instance](#)

[WriteEnPort](#)

ScanOutPort

Declares the name of a scan-out port of a Module.

Usage

Syntax 1

```
Module module_name {  
    ScanOutPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ScanOutPort port_name {  
        Source source_name ;  
        Enable enable_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of a scan-out port of a [Module](#).

A module may have zero or more scan-out ports. However, if a module has at least one [ScanInPort](#) defined, then there must be at least one ScanOutPort.

The scan-out port is used to execute a serial read (that is, not a parallel read operation). For a parallel read, use the [DataOutPort](#) port function instead.

If none of the parameters Source, Enable, or Attribute are used, you must use Syntax 1. However, in this case, the scan chain is broken at the ScanOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken scan segment will result in an error during PDL command retargeting.

Arguments

- *port_name*

A required string that identifies the name of a scan-out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ScanOutPort of the module that is also used for the respective ports of the design module.

- *Source source_name ;*

A keyword-string pair that identifies the source of the scan-out data, like a [ScanRegister](#) or a ScanOutPort of another module instance. A ScanOutPort may have at most one Source statement. The Source statement is not needed for black box modules, which are currently

not supported by Tesson. If the source _name references a [ScanRegister](#), it should reference the right most bit of the register. Further, the width of the Source must match the width of the ScanOutPort.

- Enable *enable_name* ;

A keyword-string pair that identifies the enable signal of the scan-out port. The enable signal must become active in order to connect the scan-out port with the scan source. The activation and deactivation of the enable signal is taken care of by Tesson. Please see the [OneHotScanGroup](#) to learn more about the usage of the enable signal. A ScanOutPort may have at most one Enable statement.

- Attribute *att_name = att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tesson understands several [port attributes](#) and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows the usage of the ScanOutPort and its Source statement. Observe that the width of the scan-out port is 2, hence the concatenation of the right most bits of both scan registers. Please see the discussion and examples of the [OneHotScanGroup](#) to learn more about the usage of the ScanOutPort's enable signal.

```
Module tdr2x {
    ScanInPort      si[1:0];
    ScanOutPort     so[1:0] { Source R0[0],R1[0]; }
    ShiftEnPort    se;
    SelectPort      en;
    CaptureEnPort  ce;
    UpdateEnPort   ue;
    TCKPort         tck;
    ScanInterface chain2x {
        Port se;
        Chain c0 { Port si[0]; Port so[0]; }
        Chain c1 { Port si[1]; Port so[1]; }
    }

    ScanRegister R0[9:0] {
        ScanInSource    si[0];
        ResetValue      10'b0 ;
    }
    ScanRegister R1[4:0] {
        ScanInSource    si[1];
        ResetValue      5'b0 ;
    }
}
```

Related Topics

[ScanInPort](#)

[DataOutPort](#)

[ScanInterface](#)

[OneHotScanGroup](#)

DataOutPort

Declares the name of the data out port of a Module.

Usage

Syntax 1

```
Module module_name {  
    DataOutPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    DataOutPort port_name {  
        Source source_name ;  
        Enable enable_name ;  
        RefEnum enum_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the data out port of a [Module](#).

A module may have zero or more data out ports. The number and bus-width of all declared data out ports is not correlated to the number and bus-width of any [DataInPort](#).

A data out port is used to execute a parallel read (that is, not a scan operation) to ports of the module or to readable objects inside the module, like [DataRegister](#). If data is to be scanned-out, use the [ScanOutPort](#) port function instead. If a parallel read from ports or objects of the module is to be allowed, the DataOutPort function(s) must be declared for the module. If a module does not have DataOutPort port functions, no parallel data read is allowed. Certain additional restrictions apply as explained next.

If the data output port signal has [SelectPort](#) or [ToSelectPort](#), which is part of an [ScanInterface](#) in its fanin, then it may not be target of an [iWrite](#) or [iRead](#) operation. This ensures that the scan selection remains under control of the software application.

If a data out port is connected to an [Instance](#) or [DataRegister](#) with an [AddressValue](#) statement, all data out ports are concatenated in the order of declaration until the required data width is achieved. See the discussion of [AddressValue](#) for further explanation and an example.

If none of the parameters [Source](#), [Enable](#), [Attribute](#), or [RefEnum](#) are used, you must use Syntax 1. However, in this case, the data connection is broken at the DataOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken data connection will result in an error during PDL command retargeting.

Arguments

- *port_name*

A required string that identifies the name of a data out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL DataOutPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name* ;

An optional keyword-string pair that identifies the source of the parallel output data, like a [DataRegister](#) or a [DataOutPort](#) of another module instance. A DataOutPort may have at most one Source statement. Further, the width of the Source must match the width of the DataOutPort.

- Enable *enable_name* ;

An optional keyword-string pair that identifies the enable signal of the data out port. The enable signal must become active in order to connect the data output port with the data source. The activation and deactivation of the enable signal is taken care of by Tessent. Please see the [to learn more about the usage of the enable signal](#). A DataOutPort may have at most one Enable statement.

- RefEnum *enum_name* ;

An optional keyword-string pair that references the name of a Related Topics table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. It must be understood that the enumeration table is not necessarily a complete list. It is still allowed to write to the data in ports any value, including those not listed in the enumeration table. A DataOutPort may have at most one RefEnum statement.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [icl_pin](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how the data in port is used inside the [ScanRegister](#). It is declared as the source of the capture values of the scan register, which means that the scan register will capture the value from the data in ports fq on the positive edge of the [CaptureEnPort](#) ce, under the condition that the [SelectPort](#) en is also high. Finally, observe the usage of the RefEnum and Related Topics statements.

```
Module tdr_ref {
    ScanInPort      si;
    ScanOutPort     so      {  Source R[0];   }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    DataInPort      fq[7:0]  {  RefEnum OptionalValues;  }
    DataOutPort     td[7:0]  {  Source R[7:0]; RefEnum OptionalValues;  }
    ScanRegister R[7:0] {
        CaptureSource fq[7:0];
        ScanInSource  si;
        RefEnum       OptionalValues ;
    }   Enum OptionalValues {
        reset = 8'b0 ;
        set   = 8'b11111111 ;
        chain = 8'b11001100 ;
        green = 8'hee ;
        blue  = 8'd198 ;
    }
}
```

Related Topics

[DataInPort](#)

[ScanInPort](#)

[ScanOutPort](#)

[Enum](#)

ToShiftEnPort

Declares the name of the to-shift enable port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToShiftEnPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToShiftEnPort port_name {  
        Source source_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-shift enable port of a [Module](#)'s host [ScanInterface](#).

This port is used to enable the shift operation of another module's [ScanRegister](#) elements. It is an active high output signal. A module may have zero or more to-shift enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToShiftEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

An optional keyword-string pair that identifies the source of the to-shift enable signal. A ToShiftEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToShiftEnPort.

A to-shift enable signal pointing to a shift enable signal may evaluate to a constant 0 or a constant 1. If it evaluates to a constant 0, the scan network controlled by the to-shift enable signal is considered disabled. No read or write operation into this scan chain segment may occur. If it evaluates to a constant 1, it is considered an error.

Note

 The features associated with the Source specification are currently not supported.
Tessent currently ignores the Source specification..

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several **icl_port** attributes and **icl_pin** attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```
Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux; }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn1   {  Source sel1;  }
    ScanInPort   fromTdr1;
    ToSelectPort tapEn1   {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;   ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }   ScanInterface hostTAP {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }   Instance fsm_i Of fsm  {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }   ScanRegister IR[2:0] {
        CaptureSource 3'b001;
        ResetValue    3'b000;
        ScanInSource  tdi;
    }

    ScanRegister bypass {
        CaptureSource 1'b0;
        ScanInSource  tdi;
    }   ScanMux IRMux SelectedBy fsm_i.irSel {
        1'b0 : DRMux;
        1'b1 : IR[0];
    }   ScanMux DRMux SelectedBy IR[2:0] {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    }   LogicSignal sel1 {  IR[2:0] == 3'bx01;  }
}Module fsm {
    TCKPort      tck;
    TMSPort      tms;
    TRSTPort     trst;
    ToIRSelectPort irSel;
    ToResetPort  tlr;
}
```

Related Topics

[TMSPort](#)

[ScanInterface](#)

[ToIRSelectPort](#)

[ToTCKPort](#)

[ToSelectPort](#)

[ToCaptureEnPort](#)

[ToUpdateEnPort](#)

[ToResetPort](#)

ToCaptureEnPort

Declares the name of the to-capture enable output port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
    ToCaptureEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
    ToCaptureEnPort port_name {
        Source source_name ;
        Attribute att_name = att_value ;
    }
}
```

Description

A port function that declares the name of the to-capture enable output port of a [Module](#)'s host [ScanInterface](#).

This port is used to trigger the capture operation of another module's [ScanRegister](#) elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToCaptureEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

An optional keyword-string pair that identifies the source of the to-capture enable signal. A ToCaptureEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToCaptureEnPort.

The Source may point to a CaptureEnPort of the same module, a ToCaptureEnPort of an instance or a DataMux which has a capture signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the capture enable pulse is assumed to be suppressed.

Note



The suppressing of the enable signal described above is currently not supported in Tesson.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tesson understands several **icl_port** attributes and **icl_pin** attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```
Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux; }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn1   {  Source sel1;  }
    ScanInPort   fromTdr1;
    ToSelectPort tapEn1   {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;   ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }
    ScanInterface hostTAP  {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }  Instance fsm_i Of fsm  {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }  ScanRegister IR[2:0]  {
        CaptureSource 3'b001;
        ResetValue     3'b000;
        ScanInSource   tdi;
    }

    ScanRegister bypass  {
        CaptureSource 1'b0;
        ScanInSource   tdi;
    }  ScanMux IRMux SelectedBy fsm_i.irSel  {
        1'b0 : DRMux;
        1'b1 : IR[0];
    }  ScanMux DRMux SelectedBy IR[2:0]  {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    }  LogicSignal sel1 {  IR[2:0] == 3'bx01;  }
}Module fsm {
    TCKPort      tck;
    TMSPort      tms;
    TRSTPort     trst;
    ToIRSelectPort irSel;
    ToResetPort  tlr;
}
```

Related Topics

[TMSPort](#)
[ScanInterface](#)
[ToIRSelectPort](#)
[ToTCKPort](#)
[ToSelectPort](#)

[ToUpdateEnPort](#)

[ToShiftEnPort](#)

[ToResetPort](#)

ToUpdateEnPort

Declares the name of the to-update enable port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToUpdateEnPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToUpdateEnPort port_name {  
        Source source_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-update enable port of a [Module](#)'s host [ScanInterface](#).

This port is used to trigger the update operation of another module's [ScanRegister](#) elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToUpdateEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

An optional keyword-string pair that identifies the source of the to-update enable signal. A ToUpdateEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToUpdateEnPort.

The Source may point to an UpdateEnPort of the same module, a ToUpdateEnPort of an instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed.

Note



The suppressing of the enable signal described above is currently not supported in Tesson.

- Attribute *att_name* = *att_value* ;

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tesson understands several **icl_port** attributes and **icl_pin** attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```

Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux; }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn1   {  Source sel1;  }
    ScanInPort   fromTdr1;
    ToSelectPort tapEn1   {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;   ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }
    ScanInterface hostTAP  {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }   Instance fsm_i Of fsm  {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }   ScanRegister IR[2:0] {
        CaptureSource 3'b001;
        ResetValue     3'b000;
        ScanInSource   tdi;
    }

    ScanRegister bypass {
        CaptureSource 1'b0;
        ScanInSource   tdi;
    }   ScanMux IRMux SelectedBy fsm_i.irSel {
        1'b0 : DRMux;
        1'b1 : IR[0];
    }   ScanMux DRMux SelectedBy IR[2:0] {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    }   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
    TCKPort      tck;
    TMSPort      tms;
    TRSTPort     trst;
    ToIRSelectPort irSel;
    ToResetPort  tlr;
}

```

Related Topics

[TMSPort](#)

[ScanInterface](#)

[ToIRSelectPort](#)

[ToTCKPort](#)

[ToSelectPort](#)

[ToCaptureEnPort](#)

[ToShiftEnPort](#)

[ToResetPort](#)

ToResetPort

Declares the name of the to-reset output port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToResetPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToResetPort port_name {  
        Attribute att_name = att_value ;  
        Source source_name ;  
        ActivePolarity 0 | 1;  
    }  
}
```

Description

A port function that declares the name of the to-reset output port of a [Module](#)'s host [ScanInterface](#).

The to-reset port is used as the source of the [ResetPort](#) of another module. You have the option to chose the active polarity of the to-reset port.

It is important to differentiate between the active polarity, 0 or 1, and when a to-reset signal becomes active / inactive. For example, it is possible that the to-reset signal's active polarity is 0, but it is enabled if the module's input reset port turns active high (1).

If none of the parameters Source, Attribute, or ActivePolarity are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotScanGroup](#), [OneHotDataGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#)attributes and [icl_pin](#)

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ActivePolarity 0 | 1

An optional keyword-value pair, that defines if the to-reset port of the module is considered active low (value = 0), or active high (value =1). A module may have at most one ActivePolarity statement. If the ActivePolarity statement is not given, the active polarity of the to-reset port is assumed to be the 1 (high).

The activity/inactivity of the to-reset signal is given through the single [ResetPort](#) statement of the module, or, if the module does not have a [ResetPort](#) or more than one, through the global reset signal. In other words, if the reset signal is active, the to-reset signal is active as well, however, the ActivePolarity may be different.

- Source source_name

An optional keyword-string pair that identifies the source of the to-reset signal. A ToResetPort may have at most one Source statement. However, if the module has more than one [ResetPort](#) statement, then the Source must be specified. Further, the width of the Source must match the width of the ToResetPort.

The Source property can point to various sorts of drivers within the module, causing the ToResetPort to become active under different conditions (synchronous reset, asynchronous reset, enforced local reset and so on). If the Source has not been specified, then the ToResetPort may be implicitly connected to a reset signal in the current module. See chapter “[How to Model Global Reset, Local Reset and Embedded TAPs](#)” in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ToResetPort.

Examples

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. It is an error to connect a ToResetPort to a ResetPort with different polarities.

```
Module chip_reset {
    ScanInPort si ;
    ScanOutPort so { Source inst_inst.so ; } ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;
    SelectPort selC ; ResetPort re { ActivePolarity 1 ; }
    TCKPort tck ; Instance res_inst Of res {
        InputPort re = re ;
    } Instance inst_inst Of inst {
        InputPort si = si ;
        InputPort se = se;
        InputPort ce = ce ;
        InputPort ue = ue ;
        InputPort sel = selC ;
        InputPort re = res_inst.trp ;
    }
}
Module res {
    ResetPort re ;
    ToResetPort trp { ActivePolarity 0 ; }
}
Module inst {
    ScanInPort si ;
    ScanOutPort so { Source R[0] ; }
    ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;
    SelectPort sel ;
    TCKPort tck ;

    ResetPort re { ActivePolarity 0; }
    ScanRegister R[7:0] {
        ScanInSource si ;
        ResetValue 8'b0 ;
    }
}
```

Related Topics

[ScanInterface](#)
[ToTCKPort](#)
[ToUpdateEnPort](#)
[ToSelectPort](#)
[ToCaptureEnPort](#)
[ToShiftEnPort](#)

ToSelectPort

Declares the name of the to-select port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToSelectPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToSelectPort port_name {  
        Source source_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-select port of a [Module](#)'s host [ScanInterface](#).

This port is typically used to enable the scan operation in another module. It is an active high output signal. A module may have none, one, or several to-select ports.

Arguments

- *port_name*

A required string that identifies the name of a to-select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToSelectPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort.

An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort. If the to-select diagonal fans out into a ScanInterface, which is referenced by an *AllowBroadcastingOnScanInterface* property, anywhere in the hierarchy, then the Source statement is required.

Note



Currently Tessent does not support any broadcasting.

- Attribute *att_name* = *att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several **icl_port** attributes and **icl_pin** attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. Further, the SelectPort signal for the instance inst_inst is computed based on the top level DataIn signal din.

```

Module chip {
    ScanInPort si ;
    ScanOutPort so { Source inst_inst.so ; }
    ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;
    DataInPort din ;
    ResetPort re { ActivePolarity 1 ; }
    TCKPort tck ;
    Instance res_inst Of res {
        InputPort gres = din ;
        InputPort re = re ;
    } Instance inst_inst Of inst {
        InputPort si = si ;
        InputPort se = se;
        InputPort ce = ce ;
        InputPort ue = ue ;
        InputPort sel = res_inst.tsel ;
        InputPort re = res_inst.trp ;
    }
}
Module res {
    ResetPort re ;
    DataInPort gres ;
    ToResetPort trp { ActivePolarity 0 ; }
    ToSelectPort tsel { Source gres; }
}
Module inst {
    ScanInPort si ;
    ScanOutPort so { Source R[0] ; }
    ShiftEnPort se ;
    CaptureEnPort ce ;
    UpdateEnPort ue ;
    SelectPort sel ;
    TCKPort tck ;

    ResetPort re { ActivePolarity 0; }
    ScanRegister R[7:0] {
        ScanInSource si ;
        ResetValue 8'b0 ;
    }
}

```

Related Topics

- [ToShiftEnPort](#)
- [ToUpdateEnPort](#)
- [ToResetPort](#)
- [ToCaptureEnPort](#)

ToClockPort

Declares the name of a functional to-clock output port of a Module

Usage

Syntax 1

```
Module module_name {  
    ToClockPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToClockPort port_name {  
        Source source_name ;  
        FreqMultiplier pos_int ;  
        FreqDivider pos_int ;  
        DifferentialInvOf other_clock_port_name ;  
        Period pos_int unit ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of a functional to-clock output port of a [Module](#) (that is, a clock that is not a test clock).

The test clock is defined through the [ToTCKPort](#) statement. A module may have zero or more to-clock ports.

A ToClockPort serves as the source of a functional clock to another module. It has three principal features:

- Modify an existing incoming functional clock with respect to its frequency by multiplication and division of the period
- Serve as an oscillator clock generation with a defined outgoing clock frequency
- Convert from differential to single-ended clock signals and vice versa.

If none of the parameters are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name*

An optional keyword-string pair that identifies the source of the to-clock signal. A ToClockPort may have at most one Source statement. The width of the Source must match the width of the ToClockPort. The usage of Source is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqMultiplier *pos_int*

An optional keyword-positive integer number pair that identifies the multiplication factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqMultiplier statement. The usage of FreqMultiplier is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqDivider *pos_int*

An optional keyword-positive integer number pair that identifies the division factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqDivider statement. The usage of FreqDivider is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- DifferentialInvOf *other_clock_port_name*

An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module. The usage of DifferentialInvOf is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and Period, leaving the DifferentialInvOf the only parameter other than the Attribute within a ToClockPort statement.

- Period *pos_int unit*

An optional keyword-positive integer number pair, followed by an optional unit string. If used, the unit string is exactly one of these time units: s, ms, us, ns, ps. If unit is not used, the default is “ns”.

Using the Period parameter, you can create a module that does not require an incoming clock signal to generate an outgoing clock signal of a specified period. This module is like an oscillator.

The usage of Period is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and DifferentialInvOf, leaving the Period as the only parameter other than the Attribute within a ToClockPort statement.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the

language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example represents a programmable clock generation logic that selectively uses one of two incoming reference clocks and manipulates either with a selectable set of modifiers.

```

Module chip_clock {
    ScanInPort      si ;
    ScanOutPort     so { Source clockGen_inst.so ; }
    ShiftEnPort     se ;
    CaptureEnPort   ce ;
    UpdateEnPort    ue ;
    TCKPort         tck ;
    ClockPort       refClkA ;
    ClockPort       refClkB ;
    DataInPort      din[7:0] ;
    DataOutPort     dout[7:0] { Source inst_inst.out ; }
    Instance clockGen_inst Of clockGen {
        InputPort si = si ;
        InputPort refClkA = refClkA ;
        InputPort refClkB = refClkB ;
        Parameter sclkMult = 1 ;
        Parameter sclkDiv = 3 ;
    }   Instance inst_inst Of instrument {
        InputPort clk = clockGen_inst.sclk ;
        InputPort in = din ;
    }
}
Module clockGen {
    ScanInPort      si ;
    ScanOutPort     so { Source clkSelReg[0] ; }
    ClockPort       refClkA ;
    ClockPort       refClkB ;
    Parameter sclkMult = 1 ;
    Parameter sclkDiv = 1 ;
    ToClockPort sclk {
        FreqMultiplier $sclkMult ;
        FreqDivider $sclkDiv ;
        Source refSelM ;
    }
    ScanRegister clkSelReg[2:0] {
        ScanInSource si ;
        ResetValue 3'b100 ;
    }   ClockMux refSelM SelectedBy clkSelReg[2] {
        1'b1 : refClkA ;
        1'b0 : refClkB ;
    }
}
Module instrument {
    DataInPort      in[7:0];
    DataOutPort     out[7:0];
    ClockPort       clk;

    Alias enable      = in[7]           { RefEnum YesNo;      }
    Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;     }
    Alias p1[2:0]     = in[4],in[1:0];
    Alias go          = out[0]          { RefEnum PassFail; }
    Alias done         = out[1]          { RefEnum YesNo;      }
    Alias count[5:0]  = out[7:2];    Enum PassFail {
        Pass = 1'b1;
        Fail = 1'b0;
    }
    Enum YesNo {
        Yes = 1'b1;
    }
}

```

```
    No      = 1'b0;
}
Enum Modes {
    red    = 4'b0011;
    blue   = 4'b1000;
    green  = 4'b0100;
}
}
```

Related Topics

[ClockPort](#)

[TCKPort](#)

[ToTCKPort](#)

ToTCKPort

Declares the name of a test to-test clock output port of a Module.

Usage

Syntax 1

```
Module module_name {  
    ToTCKPort port_name ;  
}
```

Syntax

```
Module module_name {  
    ToTCKPort port_name {  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of a test to-test clock output port of a [Module](#) (that is, a clock that is not a functional clock).

The test clock is defined through the [ToClockPort](#) statement. A module may have zero or more to-test clock ports. If a module has several to-test clock outputs, all clocks are considered equivalent in period and polarity.

Unlike the [ToClockPort](#) statement, there are no modifiers in the ToTCKPort statement. The outgoing to-test clock signal is just a pass-through of the incoming test clock.

If no parameter is used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example modifies the example given in [ToClockPort](#). Here, a [TCKPort](#) and a [ToTCKPort](#) port was added to the module. Observe that ToTCKPort is just passing through the test clock. No manipulation of the test clock signal can happen.

```
Module clockGen {
    ScanInPort      si ;
    ScanOutPort     so { Source clkSelReg[0] ; }
    TCKPort         tck ;
    ClockPort       refClkA ;
    ClockPort       refClkB ;
    Parameter sclkMult = 1 ;
    Parameter sclkDiv = 1 ;
    ToTCKPort       ttck ;
    ToClockPort     sclk {
        FreqMultiplier $sclkMult ;
        FreqDivider   $sclkDiv ;
        Source refSelM ;
    }
    ScanRegister clkSelReg[2:0] {
        ScanInSource si ;
        ResetValue 3'b100 ;
    }   ClockMux refSelM SelectedBy clkSelReg[2] {
        1'b1 : refClkA ;
        1'b0 : refClkB ;
    }
}
```

Related Topics

[ClockPort](#)

[TCKPort](#)

[ToClockPort](#)

ToTMSPort

Declares the name of the to-TMS port of a Module's host TAP ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToTMSPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToTMSPort port_name {  
        Source source_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-TMS port of a [Module](#)'s host TAP [ScanInterface](#). This port is typically used to provide a TMS signal to an embedded TAP controller module or to module that contains an embedded TAP controller.

If Source is used, the outgoing to-TMS signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToTMSPort of the module that is also used for the respective ports of the design module.

Any module that has a ToTMSPort must also have at least one [TMSPort](#).

- Source *source_name*

An optional keyword-string pair that identifies the logic signal that is considered gating the to-TMS signal of the module. A ToTMSPort may have at most one Source statement. Source is required if the module has more than one [TMSPort](#). If the module has only one [TMSPort](#), Source is optional and if not provided, ToTMSPort then refers to the [TMSPort](#).

The Source property can point to various sorts of drivers within the module, causing the ToTMSPort either to be in its normal mode of operation or to be continuously driven with

the value '1' (which causes the downstream embedded TAP controllers to be held in "reset" state) or to be continuously driven with the value '0' (which causes the downstream embedded TAP controller to be held in "idle" state). If the Source has not been specified explicitly, then the ToTMSPort may be implicitly connected to a tms signal in the current module. See chapter "[How to Model Global Reset, Local Reset and Embedded TAPs](#)" in the *Tessent IJTAG User's Manual* for the details on embedded TAPs and the different ways of connecting a ToTMSPort.

- Attribute *att_name* = *att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[TMSPort](#)

[ToTCKPort](#)

[ToTRSTPort](#)

ToTRSTPort

Declares the name of the to-TRST port of a Module's host TAP ScanInterface.

Usage

Syntax 1

```
Module module_name {  
    ToTRSTPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToTRSTPort port_name {  
        Source source_name ;  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-TRST port of a [Module](#)'s host TAP [ScanInterface](#). This port is used to distribute the TRST signal throughout the design. The to-TRST outgoing signal is always active low (0).

A module may have zero or more to-TRST ports. If a module has several to-TRST outputs, all to-TRST signals are considered equivalent.

If Source is used, the outgoing to-TRST signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToTRSTPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

An optional keyword-string pair that identifies the logic signal that is considered gating the to-TRST signal of the module. A ToTRSTPort may have at most one Source statement, and the width of the Source must match the width of the ToTRSTPort. Source is required if the module has more than one [TRSTPort](#). If the module has only one [TRSTPort](#), Source is optional and if not provided, ToTRSTPort then refers to the [TRSTPort](#). Further, if the module does not have a [TRSTPort](#), the ToTRSTPort refers to the global TRST signal.

The Source property can point to various sorts of drivers within the module, causing the ToTRSTPort to become active under different conditions (asynchronous global reset, enforced local reset and so on). If the Source has not been specified, then the ToTRSTPort may be implicitly connected to a trst signal in the current module. See chapter “[How to Model Global Reset, Local Reset and Embedded TAPs](#)” in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ToTRSTPort.

- Attribute *att_name* = *att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[TMSPort](#)

[ToTCKPort](#)

[ToTMSPort](#)

ToIRSelectPort

Declares the name of the to-IR Select port of a Module.

Usage

Syntax 1

```
Module module_name {  
    ToIRSelectPort port_name ;  
}
```

Syntax 2

```
Module module_name {  
    ToIRSelectPort port_name {  
        Attribute att_name = att_value ;  
    }  
}
```

Description

A port function that declares the name of the to-IR Select port of a [Module](#).

This signal is used to select between the instruction register of a TAP controller and another scan register, usually a data register. The to-IR Select signal is always associated with the TMS signal, and if present, with the TRST signal.

If the parameter Attribute is not used, you must use Syntax 1.

Arguments

- *port_name*

A required string that identifies the name of a to-IR Select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

It is recommended to use a port name for the ICL ToIRSelectPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [icl_port](#) attributes and [icl_pin](#) attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

[TMSPort](#)

[TRSTPort](#)

Instance

Names an instance of an ICL module.

Usage

Syntax 1

```
Module module_name {
    Instance instance_name Of module_name {
        Attribute att_name = att_value ;
        InputPort moduleinputport_name = concat_signal ;
        Parameter parameter_name = parameter_value ;
        AddressValue number ;
    }
}
```

Syntax 2

```
Module module_name {
    Instance instance_name Of module_name ;
}
```

Description

The Instance statement names an instance of an ICL module. It also declares the address and/or all input connections of the instance to source signals of other instances or the top level module. Note that in ICL no output connections are listed in the Instance statement.

In addition to the input port connections, [Parameters](#) defined within the module definition may be overwritten. [LocalParameter](#) cannot be overwritten though.

Syntax 2 must be used if the module has no input ports and none of the other parameters is used.

Arguments

- *instance_name Of module_name*

A required string-string pair, connected with the case-sensitive keyword ‘Of’, defining the name of the instance and referencing an existing module name of which an instance is to be created. Within a module, all instance names must be unique. Also, no module may contain an instance of itself anywhere through its hierarchy.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value are defined by you, and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- InputPort *moduleinputport_name* = *concat_signal*

A repeatable, required keyword and string-value pair, connected with the ‘=’ symbol. For each input port defined in the module there must be exactly one such line in the Instance statement, however exceptions apply as explained below. *moduleinputport_name* must refer to an input port name of the referenced module. For modules without any input ports the InputPort parameter may not be used. The order of the input port declaration in the referenced module and the order of the InputPort parameters can be different.

concat_signal may reference a single signal or a concatenation of signals of the same or compatible type. The type of *concat_signal* must match the type of the input port definition in the module, or must be a compatible type. *concat_signal* may only be one of these signals: reset, scan, data, read enable, write enable, clock, TCK, shift enable, capture enable, update enable, TMS, or TRST. The width of the module’s port definition and *concat_signal* must match. For an indexed input port of the module, the InputPort parameter may be used as often as necessary to cover the entire index range in any order. However, overlapping indices or ranges are not allowed. For instances without the AddressValue parameter, the InputPort parameter may connect the module’s WriteEnPort, ReadEnPort, AddressPort, or SelectPort to data signals; that is, a data signal is a compatible source for those port functions.

Note that not all input ports of a module must be connected explicitly. For a non-handoff module; that is, for a module that is not the top level module you hand over to a third party, several connections are automatically implied, if their source in the parent module can be determined unambiguously. It is legal to list these connections through the InputPort parameter, but it is not necessary. The following connections are automatically implied for non-hand-off module instances that do not have the AddressValue statement: TCK, shift enable, capture enable, update enable.

- Parameter *parameter_name* = *parameter_value*

A repeatable, optional keyword and string-value pair, connected with the “=” symbol. The referenced *parameter_name* must have been defined in the module referenced by *module_name*; also the type of *parameter_value* must match the type used in the module for the parameter.

Using the [Parameter](#) statement again within Instance overwrites the respective parameter value set in the module definition, but only for the current instance. Another instance of the module may use another overwrite value, or may use the original value defined in the module. Through this construct, ICL can define parameterized modules, where the final parameter is defined during each instantiation of the module.

- AddressValue *number*

A required keyword and number pair, if this instance is part of the [OneHotDataGroup](#). The AddressValue parameter may not be used if the instance is not part of a [OneHotDataGroup](#). Within the [OneHotDataGroup](#), the *number* declares a part of the address that is associated with the instance. The final address of a [DataRegister](#) is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense

number is only one element of the sum, comprising the address of a data register of the [OneHotDataGroup](#).

The following connections are automatically implied for non-hand-off module instances that do have the AddressValue statement: TCK, the instance's AddressPort is assumed to be connected to the logical address signal of the parent module, the instance's WriteEnPort is assumed to be connected to the WriteEnPort of the parent module, the instance's ReadEnPort is assumed to be connected to the ReadEnPort of the parent module, and the instance's DataInPort is assumed to be connected to the logical data in signals of the parent module.

Note

In IJTAG, the Instance statement describes the “AllowBroadcastingOnScanInterface” parameter. This parameter is currently not supported in Tessent.

Examples

Example 1

The following example demonstrates the usage of automatic default connections in the common case of a scan interface. Observe that the instance declaration for tdr_inst of module tdr only declares the scan input port connection. TCK and all the enable signals are connected fully automatically to the respective signals of the parent module. Explicit connections are only necessary, if there are multiple signals of the same type in the parent module, for example two select ports.

```
Module chip_tdr {
    ScanInPort      si;
    ScanOutPort     so      { Source tdr_inst.so; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    Instance tdr_inst Of tdr {
        InputPort si = si ;
    }
}
Module tdr {
    ScanInPort      si;
    ScanOutPort     so      { Source R[0]; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource  si;
        ResetValue    8'b0 ;
    }
}
```

Example 2

The following example demonstrates not only the usage of the [OneHotDataGroup](#), but also the usage of the [InputPort](#) parameter and some of the compatible signal connections. Observe that module “top” does have neither a [ReadEnPort](#), nor a [WriteEnPort](#). Instead it has 2 [DataInPort](#) statements that are connected to the respective input ports of the instances of module “phy”. Since the connection of the data signals of the parent module (that is, top) to the instances is done implicitly, by the order of declaration, it is important that the 16 data input bits of [DataIn](#) are declared before the 2 bits of the write and read enable. Also note that the module top defines just one [AddressPort](#) of 8 bits, which is implicitly mapped to the two 4-bit [AddressPorts](#) of phy in their order of declaration. Further, observe that the [AddressPort](#) of “bank” only 2-bits wide, since only two bits are required for this module’s operation.

```

Module top {
    AddressPort address[7:0] ;
    DataInPort dataIn[15:0] ;
    DataInPort write_enable ;
    DataInPort read_enable ;
    DataOutPort dataOut[15:0] {
        Source outMux[15:0] ;
        Enable read_enable ;
    }
    OneHotDataGroup outMux[15:0] {
        Instance phy1 Of phy {
            AddressValue 2'b01, 6'b000000 ;
            InputPort we = write_enable ;
            InputPort re = read_enable ;
        }
        Instance phy2 Of phy {
            AddressValue 2'b10, 6'b000000 ;
            InputPort we = write_enable ;
            InputPort re = read_enable ;
        }
    }
}
Module phy {
    WriteEnPort 7 we ;
    ReadEnPort re ;
    AddressPort address1[3:0] ;
    AddressPort address2[3:0] ;
    DataInPort dataIn1[7:0] ;
    DataInPort dataIn2[7:0] ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ;
}
OneHotDataGroup outMux[15:0] {
    Instance bank1 Of bank {
        AddressValue 4'b0001,2'b00 ;
    }
    Instance bank2 Of bank {
        AddressValue 4'b0010,2'b00 ;
    }
}
Module bank {
    WriteEnPort we ;
    ReadEnPort re ;
    AddressPort address[1:0] ;
    DataInPort dataIn[15:0] ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ;
}
OneHotDataGroup outMux[15:0] {
//    AddressValue 2'bx,4'bx,2'b<num>
    DataRegister D0[15:0] { AddressValue 2'b00 ; }
    DataRegister D1[15:0] { AddressValue 2'b01 ; }
    DataRegister D2[15:0] { AddressValue 2'b10 ; }
    DataRegister D3[15:0] { AddressValue 2'b11 ; }
}
}

```

Related Topics

[ScanInterface](#)

[DataRegister](#)

[ScanRegister](#)

[OneHotDataGroup](#)

[OneHotScanGroup](#)

Parameter

Define a named variable and assigns a default value.

Usage

Syntax 1 : Defining a parameter

```
Module module_name {  
    Parameter parameter_name = parameter_value ;  
}
```

Syntax 2 : Optionally overwriting a parameter in an Instance statement

```
Module module_name {  
    Instance instance_name of other_module_name {  
        Parameter parameter_name = parameter_value ;  
    }  
}
```

Description

With the help of the Parameter statement you can define a named variable and assign a default value.

The default value is specified in the module that declares the Parameter. The value can be overwritten with every instance of the Module. Through this, ICL allows you to create parameterized modules.

You build a reference to the Parameter by using the ‘\$’ symbol in front of the parameter_name.

The parameter reference can be used in many locations, for example in an Enum statement. The reference will be resolved to the current value before the statement the Parameter is evaluated.

Arguments

- Parameter *parameter_name* = *parameter_value*

A required string-value pair, connected with the ‘=’ symbol. The parameter name must be unique across all Parameter and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an Instance name “inst\$size”, where “size” is a parameter

Although parameters can be used to define other parameters, a circular definition is not allowed.

Note

 Currently Tessent does not support the Callback functionality.

Examples

Example 1

The following example shows several usages of Parameter. In particular observe that the Parameter ‘Width’ is used to parameterize the values in the enumeration table.

```
Module inst {
    ScanInPort      si;
    ScanOutPort     so      { Source R[0]; }
    TCKPort         tck;
    Parameter Width = 8 ;
    Parameter Regmax = $Width - 1 ;
    Parameter userValue = 8'hff ;
    Parameter Val_A = 4'b1001, $userValue, 4'b1111 ;
    Parameter RegName = "R" ;
    LocalParameter Regmin = 0 ;

    ScanRegister R[$Regmax:$Regmin] {
        Attribute Register = "$RegName($RegMax:$RegMin)" ;
        ScanInSource si;
        ResetValue   Reset ;
        RefEnum      SomeValues ;
    }   Enum SomeValues {
        Reset = $Width'b0 ;
        User1 = $userValue ;
    }
}
```

Example 2

The following example uses the module defined above. In its instantiation, the parameters ‘Width’ and ‘userValue’ will be overwritten. This allows ICL to create parameterized modules, where the size of registers will be fixed during the instantiation of the module and not during the declaration of the module.

```
Module top {
    ScanInPort      si;
    ScanOutPort     so      {  Source inst2.so;    }
    ShiftEnPort     se;
    SelectPort       en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;

    Instance inst1 Of inst {
        InputPort si = si ;
        Parameter Width = 16 ;
        Parameter userValue = 16'hffff ;
    }
    Instance inst2 Of inst {
        InputPort si = inst1.so ;
        Parameter Width = 4 ;
        Parameter userValue = 4'ha ;
    }
}
```

Related Topics

[LocalParameter](#)

[Instance](#)

[Alias](#)

[Enum](#)

LocalParameter

Has the same semantic and use model as the Parameter statement but cannot be overwritten in the Instance statement.

Usage

```
Module module_name {  
    LocalParameter parameter_name = parameter_value ;  
}
```

Description

A LocalParameter has the same semantic and use model as the [Parameter](#) statement. The only difference is that a LocalParameter cannot be overwritten in the [Instance](#) statement.

Arguments

- *parameter_name = parameter_value*

A required string-value pair, connected with the ‘=’ symbol. The parameter name must be unique across all [Parameter](#) and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an [Instance](#) name “inst\$size”, where “size” is a parameter

Although parameters can be used to define other parameters, a circular definition is not allowed.

Related Topics

[Parameter](#)

[Instance](#)

[Alias](#)

[Enum](#)

Enum

Adds an entry to an enumeration table that links a mnemonic string to an arbitrary value.

Usage

```
Module module_name {
    Enum enum_name {
        name = value ;
    }
}
```

Description

With the **Enum** statement, ICL introduces an enumeration table that links a mnemonic string to an arbitrary value. This string can then be used in lieu of the value at places like [DataInPort](#), [DataOutPort](#), and [LogicSignal](#). For the [ScanRegister](#) and the [DataRegister](#) the string can be used for the DefaultLoadValue, ResetValue, and CaptureSource ([ScanRegister](#) only).

Further, PDL commands like [iWrite](#) or [iRead](#) can also use the name instead of the value. Choosing meaningful mnemonics can make both, ICL and PDL, much more human readable.

In no usage of the enumeration table are you limited to only the names listed in the table. You are free to continue using any legal value, and in particular values not listed in the table. The example below shows such a case.

Arguments

- *enum_name*

A required string that identifies the name of the enumeration table. Using this name the RefEnum parameter inside the [ScanRegister](#) statement, for example, can reference the enumeration table. The *enum_name* must be unique among all **Enum** statements within the module.

- *name = value*

A required, repeatable string-value pair, connected with the '=' symbol. These pairs make up the enumeration table. All values must be of the same width, and the names must be unique within the same **Enum**. An enumeration table must contain at least one name-value pair.

Examples

The following enumeration table shows several examples of value definitions. The width of the value depends on the usage of the table entries in other objects, like [ScanRegister](#) or [DataInPort](#), referenced by RefEnum. within these statements. The number type is not relevant as shown. Also observe that two [Parameter](#) were defined, which are used in the enumeration table 'Modes'. As usual, these parameters could be overwritten differently in each instance of the module.

```
Module instrument {
    DataInPort  in[7:0];
    DataOutPort out[7:0];
    ClockPort   clk;

    Alias enable      = in[7]           { RefEnum YesNo;      }
    Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;     }
    Alias p1[2:0]     = in[4],in[1:0];
    Alias go         = out[0]          { RefEnum PassFail;  }
    Alias done        = out[1]          { RefEnum YesNo;      }
    Alias count[5:0]  = out[7:2];
    Parameter userEnum1val = 4'b1111 ;
    Parameter userEnum2val = 4'h22 ;
    Enum PassFail {
        Pass = 1'b1;
        Fail = 1'b0;
    }
    Enum YesNo {
        Yes = 1'b1;
        No = 1'b0;
    }
    Enum Modes {
        none   = 4'b0 ;
        red    = 4'b0011 ;
        blue   = 4'haa ;
        green  = 4'd9 ;
        userEnum1 = $userEnum1val ;
        userEnum2 = $userEnum2val ;
    }
}
```

Next, here is a snippet of a dofile, writing to the ‘mode’ [Alias](#) of the module, using the names defined in the ‘Mode’ enumeration table which is associated with the alias through the [RefEnum](#) parameter. The done [Alias](#) is then read from ‘out’.

```
open_pattern_set enumTest
iWrite mode red
iApply
iWrite mode 0b1010
iApply
iRead done Yes
iApply
close_pattern_set
write_patterns enumTest.pdl -pdl
```

The following lists the contents of the saved PDL file enumTest.pdl:

```
iProcsForModule instrument
iProc enumTest {} {
    iWrite in[6] 0b0
    iWrite in[5] 0b0
    iWrite in[3] 0b1
    iWrite in[2] 0b1
    iApply
    iWrite in[6] 0b1
    iWrite in[5] 0b0
    iWrite in[3] 0b1
    iWrite in[2] 0b0
    iApply
    iRead out[1] 0b1
    iApply
}
```

Related Topics

[iRead](#)
[LogicSignal](#)
[ScanRegister](#)
[iWrite](#)
[DataInPort](#)
[DataOutPort](#)
[DataRegister](#)

Alias

Collects signals from different origins to form a named, new, virtual signal.

Usage

Syntax 1

```
Module module_name {
    Alias alias_name = concat_signal ;
}
```

Syntax 2

```
Module module_name {
    Alias alias_name = concat_signal {
        RefEnum enum_name ;
        AccessTogether ;
        iApplyEndState value ;
    }
}
```

Description

The Alias statement collects signals from different origins to form a named, new, virtual signal.

The name of the alias can be used in any operation or construct where the original signals would have been allowed. For example, PDL commands like [iWrite](#) or [iRead](#) can use the alias name. Choosing meaningful alias names can make both ICL and PDL, much more human readable.

A second usage of the Alias statement is to instruct Tesson to automatically return to a specified value for the selected signals. Typically this feature is used to return a write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

If none of the parameters RefEnum, AccessTogether, or iApplyEndState are used, you must use Syntax 1.

The name of the Alias must be unique among all port names, and among all the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#).

Arguments

- `alias_name = concat_signal`

A required string-string pair, connected with the '=' symbol. The `alias_name` defines the name of the alias, whereas `concat_signal` refers to one signal, or a concatenation of signals. Such a concatenation of signals is created by a signal name followed by a comma (','), followed by another signal name and so on. The concatenated signals can only be from [DataInPort](#), [ScanInPort](#), [DataOutPort](#), or [ScanOutPort](#). The width of the alias name should match the width of the concatenated signal.

An alias name can become part of another Alias statement, however circular definitions are not allowed. Further, if any of the signals in the concatenation has more than one level of hierarchy; that is, the hierarchical instance-port path name contains the dot (“.”), the Alias may not be used in any other ICL statements, but only in PDL.

- RefEnum *enum_name*

An optional keyword-string pair that references the name of a Related Topics table. Using the enumeration table, the writing to the aliased signal can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. You are still allowed to read or write any value, including those not listed in the enumeration table. An Alias statement may have at most one RefEnum parameter.

- AccessTogether

An optional keyword that requests Tessent to operate all signals listed in *concat_signal* at the same time. If this turns out to be not possible, for example, based on mutually exclusive hardware requirements, an error will be returned. Without this keyword, Tessent is allowed to find a solution where some of the signals are operated first, followed by operating another set of signals in the *concat_signal* Alias.

Note



Currently Tessent does not support the AccessTogether parameter.

- iApplyEndState *value*

An optional keyword-string pair that identifies the value to which the signals listed *concat_signal* will return to automatically, if ever changed by an [iWrite](#) command. The value will return with the next [iApply](#) PDL command. Typically this feature is used to return the write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

There are additional conditions to the signals and value used in the Alias and iApplyEndState statement and parameter: All resolved signals must be legal targets of the [iWrite](#) command; *value* may not contain unknown values; if a signal is part of multiple iApplyEndState parameters, all specified values must be the same; if the signals resolve to register bits, either [ScanRegister](#) or [DataRegister](#), the register must have a ResetValue parameter defined and the reset value should match the iApplyEndState-value for these signals; if a signal does not resolve to register bits, it must resolve to primary inputs, no other resolved signal source is allowed for the *concat_signal*, if the iApplyEndState parameter is used; further, there may not be a [DataOutPort](#) with an Enable parameter between the resolved signal source and the signal specified in *concat_signal of the Alias statement*.

Examples

The following example shows several usages of the Alias statement.

```
Module instrument {
    DataInPort  in[8:0];
    DataOutPort out[7:0];
    ClockPort   clk;

    Alias write_en      = in[8]{ RefEnum YesNo; iApplyEndState 1'b0; }
    Alias enable        = in[7]           { RefEnum YesNo;          }
    Alias mode[3:0]     = in[6:5],in[3:2] { RefEnum Modes;         }
    Alias p1[2:0]       = in[4],in[1:0];
    Alias go            = out[0]          { RefEnum PassFail;      }
    Alias done           = out[1]          { RefEnum YesNo;         }
    Alias count[5:0]    = out[7:2];
    Enum PassFail {
        Pass = 1'b1;
        Fail = 1'b0;
    }
    Enum YesNo {
        Yes = 1'b1;
        No = 1'b0;
    }
    Enum Modes {
        none = 4'b0 ;
        red  = 4'b0011 ;
        blue = 4'haa ;
        green = 4'd9 ;
    }
}
```

Related Topics

[iRead](#)
[LogicSignal](#)
[ScanRegister](#)
[iWrite](#)
[DataInPort](#)
[DataOutPort](#)
[DataRegister](#)

LogicSignal

Creates a new signal that can then be referenced in other ICL statements.

Usage

```
Module module_name {
    LogicSignal signal_name {
        expression ;
    }
}
```

Description

The LogicSignal statement creates a new signal named *signal_name*, which itself can then be referenced in other ICL statements, and in particular other LogicSignal statements.

Using the LogicSignal statement, logical operations between signals like AND or EXOR can be implemented.

Arguments

- *signal_name*

A required string that identifies the newly created signal name. The value it assumes depends on the result of the evaluation of *expression*.

The name of the LogicSignal must be unique among all port names, and among all the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [Alias](#).

In any case, the width of *signal_name* is always 1. In particular, comparisons and Boolean logic operations return 1'b1, if the expression evaluates to true, and 1'b0, if the expression evaluates to false.

- *expression*

A required string that defines a Boolean expression that evaluates to a single-bit result, based on the signals and their value used in the expression.

The common Boolean and Logical operators are supported. In decreasing order of preference, these are: Boolean not (!), Bitwise not (~), logical AND (&&) and logical OR(||), bitwise AND (&), bitwise OR (), and bitwise XOR (^), logical equality (==) and inequality (!=).

All operators are left-to-right associative, and operators with equal precedence are evaluated left-to-right. Associativity and precedence may be expressly specified with parentheses () .

Valid operands are registers, ports, sized and unsized numbers, logical signals, as well as parameters and aliases that resolve into those. In addition, signals can be concatenated.

For expressions containing unsized numbers, it must be possible to determine the required width of the unsized number unambiguously. If the width must be extended, the additional

bits will either be filled with 0, if the most significant bit is either 0 or 1, or the bits will be filled with unknown (x).

For the logical equality and inequality operation, the width of the operands on either side of the operator must be the same.

Any parameter reference will be resolved before the expression is evaluated.

Enumeration table entries can be used for operands which are used in their entirety in the expression, and not just a part of the operand. The operand must have declared its binding to this enumeration by means of the RefEnum statement. Enumeration values are allowed only in combination with equality (==) and inequality (!=) operators. See the example below.

Examples

The following example shows several usages of the LogicSignal statement. sig1 demonstrates the usage of an enumeration table value entry. Note that ‘mode[3:0]’ is the entire operand for which the Enum ‘Modes’ applies. Therefore, any table entry may be used for this operand. sig2 shows the usage of concatenation, the expansion of an unsigned number and the usage of a parameter reference. Observe that the width of the right operand is 5, whereas the width of the sized elements of the left operand is 3. Therefore, the unsigned number ‘0’ will be expanded to width 2, matching the width of the right operand. Lastly, sig3 shows an example of bit-operations.

```
Module instrument {
    [ ... ]
    DataInPort  in[7:0];
    [ ... ]
    Alias enable      = in[7]           { RefEnum YesNo;      }
    Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;   }
    Parameter user1 = 4'b1010;
    LogicSignal sig1 { (mode==red) && (enable==1'b1) ; }
    LogicSignal sig2 {  in[4],1'b1,in[0],0 == 1'b0,$user1 ; }
    LogicSignal sig3 { (in[6:5] | in[3:2]) != 2'b10 ; }
    Enum Modes {
        none      = 4'b0 ;
        red       = 4'b0011 ;
        blue      = 4'haa ;
        green     = 4'd9 ;
    }
    [ ... ]
}
```

ScanRegister

Defines and names a scan register of a certain width and with certain properties.

Usage

```
Module module_name {
    ScanRegister register_name {
        Attribute att_name = att_value;
        ScanInSource scan_in_source;
        CaptureSource capture_source;
        DefaultLoadValue default_load_value;
        ResetValue reset_value;
        RefEnum enum_name;
    }
}
```

Description

The statement ScanRegister defines and names a scan register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the scan register.

Arguments

- *register_name*

A required string that identifies the name of a the scan register of the module. The name of the ScanRegister must be unique among all port names, and among all the following objects: [DataRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored by any IJTAG software tool. Tessonnt understands several [port attributes](#) and [pin attributes](#)as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ScanInSource *scan_in_source*

A required keyword-string pair that identifies the signal that is providing the scan data to the left most bit of the scan chain. If *scan_in_source* references another scan register, it must reference the right-most bit of that scan register (that is, the ‘scan out bit’).

For ranged scan registers; that is, scan registers that have a width > 1, the scan register always shifts from left to right, independently if the left most index is larger than the right most index, or vice-versa.

- CaptureSource *capture_source*

An optional keyword-string pair that identifies the signal from which the ScanRegister captures data in response to the [CaptureEnPort](#) and [TCKPort](#) signal. If used, the width of *capture_source* must match the width of the register.

- DefaultLoadValue *default_load_value*

An optional keyword-string pair that identifies the value that each scan register cell will assume if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- ResetValue *reset_value*

An optional keyword-string pair that identifies the value that each scan register cell will assume as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the scan register does not respond to the module's incoming reset signal. However, if the module contains scan registers with a ResetValue parameter, the module must have only one ResetPort statement.

When both the DefaultLoadValue parameter the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- RefEnum *enum_name*

An optional keyword-string pair that references the name of an table. Using the enumeration table, reading and writing to the scan register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A ScanRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the scan register.

Additional Rules

A ScanRegister is considered to be in the active scan path if it responds to the [ShiftEnPort](#) signal associated to the ScanInterface through an active ScanInterface. It holds its data as long as the ShiftEnPort remains inactive (low) between the capture- and the update-cycle.

Any scan register bit that controls parts of a scan network (that is, the select line of a ScanMux, a SelectPort or ToSelectPort), or is part of a ScanInterface, anywhere in the hierarchy of the design, will have a non-x reset value (that is, a reset value of 0 or 1).

If a scan register bit is the source of a data signal, the scan register must have an update stage that (or an equivalent implementation) which changes only in response to an active UpdateEnPort signal.

It is important to understand the rules which determine what value gets loaded into the scan register bits. Assume no user-defined care bits have ever been written to any scan register bits. Now an iApply block accesses the ScanRegister. The scan register bits will assume a value in this order of preference: The specified values required to satisfy the operations in the iApply block; the value DefaultLoadValue specifies, if is a 0 or 1; the value ResetValue specifies, if it is a 0 or 1; any scan register bits still unassigned a 0 or 1, will be loaded with a 0.

If a scan register receives a reset signal, all scan register bits assume the value specified in *reset_value*. For the remaining x-values apply the sequence of events outlined in the paragraph above.

If a ScanRegister is on the active scan path (that is, a shift operation may happen), any scan register bits that are not required to satisfy the operations in the current iApply block will remain unchanged from their current value.

Examples

Example 1

The following shows a complete example ICL module definition of a TAP controller. Observe that the two different ScanRegisters: ‘IR’ has a width of 3, whereas ‘bypass’ has a width of 1. ‘IR’ must have as ResetValue parameter defined, since it controls the [ScanMux](#) DRMux. Further, observe the usage of the name of the scan register in the expression of the LogicSignal ‘sel1’.

```

Module tap {
    TCKPort      tck;
    ScanInPort   tdi;
    ScanOutPort  tdo      {  Source IRMux; }
    TMSPort      tms;
    TRSTPort     trst;
    ToSelectPort tdrEn1   {  Source sel1;  }
    ScanInPort   fromTdr1;
    ToSelectPort tapEn1   {  Source IR[2];  }
    ToCaptureEnPort tce;
    ToShiftEnPort tse;
    ToUpdateEnPort tue;
    ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
    }
    ScanInterface hostTAP  {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
    }
    Instance fsm_i Of fsm  {
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
    }
    ScanRegister IR[2:0] {
        CaptureSource 3'b001;
        ResetValue     3'b000;
        ScanInSource   tdi;
    }
    ScanRegister bypass {
        CaptureSource 1'b0;
        ScanInSource   tdi;
    }
    ScanMux IRMux SelectedBy fsm_i.irSel {
        1'b0 : DRMux;
        1'b1 : IR[0];
    }
    ScanMux DRMux SelectedBy IR[2:0] {
        3'bx00 : bypass;
        3'bx01 : fromTdr1;
    }
    LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}
Module fsm {
    TCKPort      tck;
    TMSPort      tms;
    TRSTPort     trst;
    ToIRSelectPort irSel;
    ToResetPort   tlr;
}

```

Example 2

The following example shows an ICL module definition of a sensor where CaptureSource can be used to combine multiple DataInPorts to form the width of the ScanRegister.

```
Module sensor {
    TCKPort      tck;
    TRSTPort     rst;
    ScanInPort   si;
    ScanOutPort  so;      {  Source  S_reg[0]; }
    CaptureEnPort capture_en;
    ShiftEnPort  shift_en;
    UpdateEnPort update_en;
    SelectPort   sel_en;
    DataInPort   sdata_in [2:0];
    DataInPort   sens_from;
    DataInPort   cntrl [1:0];
    ScanRegister S_reg [5:0] {
        ResetValue  6'b000000;
        CaptureSource sdata_in, sens_from, cntrl ;
        ScanInSource si;
    }
}
```

Related Topics

[ScanInPort](#)

[DataOutPort](#)

[iApply](#)

[DataRegister](#)

[ShiftEnPort](#)

DataRegister

Defines and names a data register of a certain width and with certain properties.

Usage

Syntax 1

```
Module module_name {
    DataRegister register_name {
        WriteDataSource concat_signal ;
        WriteEnSource enable_source ;
        ResetValue reset_value ;
        DefaultLoadValue load_value ;
        RefEnum enum_name ;
        Attribute att_name = att_value;
    }
}
```

Syntax 2

```
Module module_name {
    DataRegister register_name {
        AddressValue address_value ;

        ResetValue reset_value ;
        DefaultLoadValue load_value ;
        RefEnum enum_name ;
        Attribute att_name = att_value;
    }
}
```

Description

The statement DataRegister defines and names a data register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the data register. The DataRegister is used for parallel reading and writing as opposed to a [ScanRegister](#), which offers a scan-based access.

There are three, mutually exclusive types of DataRegisters: A Selectable DataRegister, for which a signal must be defined from which the DataRegister captures data; an Addressable DataRegister, for which an AddressValue must be specified; and a CallBack DataRegister. You should use Syntax 1 if you have a Selectable DataRegister. You should use Syntax 2 if you have an Addressable DataRegister.

Further, the Addressable DataRegister can only be used inside a [OneHotDataGroup](#), and inside a [OneHotDataGroup](#) there can only be Addressable DataRegisters.

Note



Currently Tessent does not support the Callback functionality.

Arguments

- *register_name*

A required string that identifies the name of a data register of the module. The name of the DataRegister must be unique among all port names, and among all the following objects: [ScanRegister](#), [ScanMux](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

- *ResetValue reset_value*

An optional keyword-string pair that identifies the value that each data register cell will assume as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the data register does not respond to the module's incoming reset signal. However, if the module contains data registers with a ResetValue parameter, the module must have only one ResetPort statement.

When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- *DefaultLoadValue default_load_value*

An optional keyword-string pair that identifies the value that each data register cell will assume if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- *RefEnum enum_name*

An optional keyword-string pair that references the name of an [Enum](#) table. Using the enumeration table, reading and writing to the data register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A DataRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the data register.

- Attribute *att_name = att_value*

A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored by any IJTAG software tool. Tessent understands several [port attributes](#) and [pin_attributes](#) as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- WriteDataSource *concat_signal*

A keyword and string pair required for a selectable DataRegister. *concat_signal* references a single signal or a concatenation of signals from where the DataRegister will capture data in response to a write request when WriteEnSource turns active (high). *concat_signal* must refer either to the module's [DataInPorts](#), another DataRegister, or to a [ScanRegister](#).

- WriteEnSource *enable_source*

A keyword and string pair required for a selectable DataRegister. *enable_source* references a 1-bit wide signal which determines when the DataRegister can capture data from WriteDataSource. The DataRegister will capture data in response to a write request when WriteEnSource turns active (high).

- AddressValue *address_value*

A keyword and number pair required if an Addressable DataRegister; that is, the DataRegister is part of the [OneHotDataGroup](#). The AddressValue parameter may not be used if the DataRegister if not part of a [OneHotDataGroup](#). Within the [OneHotDataGroup](#), the *number* declares a part of the address that is associated with the DataRegister. The final address of a [DataRegister](#) is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense *number* is only one element of the sum, comprising the address of a data register of the [OneHotDataGroup](#).

The DataRegister will be loaded from the logical data-in signal with the right most indices first, when the DataRegister's address has been applied and the module's WriteEnPort is active (high). The address of a DataRegister is computed by adding all address_values of all [Instances](#) leading to the DataRegister.

Similarly, reading from the Addressable DataRegister requires the module's ReadEnPort being active (high), and the address of the DataRegister being applied.

Examples

The following example shows a selectable data register. An example of addressable data registers can be found with the [OneHotDataGroup](#) statement, as well as with the [Instance](#) statement.

Observe the usage of the [Parameter](#) statement as well as Parameter overwrites. Further observe how the write enable signal is computed for the data registers in module bank using a [LogicSignal](#) statement and concatenation of the parameterized signals of the comparisons.

All together, this example shows how a selectable DataRegister can be used very much in a way of an addressable DataRegister, but without any of the constraints that come with an addressable DataRegister. In this particular implementation, the read and write enable signal of bank are explicitly managed (see for example, the usage of [Dx_Mux](#)). Depending on your DataRegister implementation, you may be able to relax this and allow reading from and writing to a data register simultaneously. Tesson sets up constraints for the IJTAG software such that no race conditions can occur.

```

Module chip_selectable_dataregister {
    DataInPort write_enable ;
    DataInPort read_enable ;
    DataInPort address[7:0] ;
    DataInPort dataIn[15:0] ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ; }
    Parameter bank1Address = 4'b0011 ;
    Parameter bank2Address = 4'b1100 ;
    DataMux outMux[15:0] SelectedBy address[7:4] {
        $bank1Address : bank1.dataOut[15:0] ;
        $bank2Address : bank2.dataOut[15:0] ;
    }
    Instance bank1 Of bank {
        InputPort we = write_enable ;
        InputPort re = read_enable ;
        InputPort dataIn[15:0] = dataIn[15:0] ;
        InputPort address[3:0] = address[3:0] ;
    }
    Instance bank2 Of bank {
        InputPort we = write_enable ;
        InputPort re = read_enable ;
        InputPort dataIn[15:0] = dataIn[15:0] ;
        InputPort address[3:0] = address[3:0] ;
        Parameter D0_Address = 4'b1000 ;
        Parameter D1_Address = 4'b1100 ;
    }
}
Module bank {
    DataInPort we ; Alias weR = we { iApplyEndState 1'b0 ; }
    DataInPort re ;
    DataInPort address[3:0] ;
    DataInPort dataIn[15:0] ;
    DataOutPort dataOut[15:0] { Source Dx_Mux[15:0] ;
Enable re; } Parameter D0_Address = 4'b0000 ;
    Parameter D1_Address = 4'b0001 ;
    DataMux Dx_Mux[15:0] SelectedBy we,address[3:0] {
        1'b0, $D0_Address : D0[15:0] ;
        1'b0, $D1_Address : D1[15:0] ;
    }
    LogicSignal D0_wen { re,we,address[3:0] == 2'b01, $D0_Address ; }
    DataRegister D0[15:0] {
        ResetValue 16'b0 ;
        WriteEnSource D0_wen ;
        WriteDataSource dataIn[15:0] ;
    }
    LogicSignal D1_wen { re,we,address[3:0] == 2'b01, $D1_Address ; }
    DataRegister D1[15:0] {
        ResetValue 16'b0 ;
        WriteEnSource D1_wen ;
        WriteDataSource dataIn[15:0] ;
    }
}

```

Related Topics

[DataInPort](#)

[DataOutPort](#)

[ScanRegister](#)

[OneHotDataGroup](#)

ScanMux

Creates and names a scan multiplexer.

Usage

```
Module module_name {
    ScanMux mux_name SelectedBy mux_select_signal {
        mux_select_value : scan_source_name ;
    }
}
```

Description

The ScanMux statement creates and names a scan multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. No signal types may be mixed for any of the multiplexers. The select signal of the multiplexers may be sourced only by data signals. Note that the parallel output of a scan register is also a data signal.

A common design structure is to have a ScanMux, with a 1-bit TDR at the output of the ScanMux, and the parallel output of the TDR connected to the select input of the mux. In this configuration, the TDR behind the mux is always on the scan path, independent of which scan input ports of the mux have been selected. Through this TDR, Tesson can select either of the scan input ports of the mux as needed.

Arguments

- *mux_name*

A string that identifies the name of a scan multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: [ScanRegister](#), [DataRegister](#), [DataMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

Using this name, the outgoing signal of the multiplexer can be referenced. The width of *mux_name* must match the width of the *scan_source_name*.

- *mux_select_value : scan_source_name*

A repeatable string-string pair, connected by a colon symbol ‘:’. A ScanMux must have at least one *mux_select_value - scan_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *scan_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Examples

The following shows usages of all the multiplexer types.

```
Module chip2 {
    ScanInPort siA ;
    ScanInPort siB ;
    ScanOutPort so      { Source scanM ; }
    ShiftEnPort sel ;
    CaptureEnPort cel ;
    UpdateEnPort uel ;
    ShiftEnPort se2 ;
    CaptureEnPort ce2 ;
    UpdateEnPort ue2 ;
    TCKPort tck ;
    ClockPort clkA ;
    ClockPort clkB ;
    ToClockPort clk { Source clockM ; }
    DataInPort dselA ;
    DataInPort dselB[2:0] ;
    DataOutPort do[3:0] { Source dataM[3:0] ; }
    ScanInterface cA12 {
        Port sel; Port cel; Port uel;
        Chain c1 { Port siA; Port so ; }
    }
    ScanInterface cB12 {
        Port se2; Port ce2; Port ue2;
        Chain c1 { Port siB; Port so ; }
    }
    ScanRegister regA[7:0] {
        ScanInSource siA ;
        ResetValue 8'b0 ;
    }
    ScanRegister regB[7:0] {
        ScanInSource siB ;
        ResetValue 8'b0 ;
    }
    ScanMux scanM SelectedBy dselA {
        1'b0 : regA[0] ;
        1'b1 : regB[0] ;
    }
    ClockMux clockM SelectedBy dselA {
        1'b0 : clkA ;
        1'b1 : clkB ;
    }
    DataMux dataM[3:0] SelectedBy dselB[2:0] {
        3'bx00 : regA[7:4] ;
        3'bx01 : regA[3:0] ;
        3'bx1x : regB[5:2] ;
    }
}
```

Related Topics

[DataMux](#)

[ClockMux](#)

DataMux

Creates and names a data multiplexer.

Usage

```
Module module_name {
    DataMux mux_name SelectedBy mux_select_signal {
        mux_select_value : data_source_name ;
    }
}
```

Description

The DataMux statement creates and names a data multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. No signal types may be mixed for any of the multiplexers. The select signal of the multiplexers may be sourced only by data signals.

A DataMux can also be used for the gating of reset signals, tms signals and trst signals. See chapter “[How to Model Global Reset, Local Reset and Embedded TAPs](#)” in the *Tessent IJTAG User's Manual* for the details about the reset, trst and tms gating.

Arguments

- *mux_name*

A string that identifies the name of a data multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [ClockMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

Using this name, the outgoing signal of the multiplexer can be referenced. The width of *mux_name* must match the width of the *data_source_name*.

- *mux_select_value : data_source_name*

A repeatable string-string pair, connected by a colon symbol ‘:’. A DataMux must have at least one *mux_select_value* - *data_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *data_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Examples

The following shows usages of all the multiplexer types.

```
Module chip2 {
    ScanInPort siA ;
    ScanInPort siB ;
    ScanOutPort so      { Source scanM ; }
    ShiftEnPort sel ;
    CaptureEnPort cel ;
    UpdateEnPort uel ;
    ShiftEnPort se2 ;
    CaptureEnPort ce2 ;
    UpdateEnPort ue2 ;
    TCKPort tck ;
    ClockPort clkA ;
    ClockPort clkB ;
    ToClockPort clk { Source clockM ; }
    DataInPort dselA ;
    DataInPort dselB[2:0] ;
    DataOutPort do[3:0] { Source dataM[3:0] ; }
    ScanInterface cA12 {
        Port sel; Port cel; Port uel;
        Chain c1 { Port siA; Port so ; }
    }
    ScanInterface cB12 {
        Port se2; Port ce2; Port ue2;
        Chain c1 { Port siB; Port so ; }
    }
    ScanRegister regA[7:0] {
        ScanInSource siA ;
        ResetValue 8'b0 ;
    }
    ScanRegister regB[7:0] {
        ScanInSource siB ;
        ResetValue 8'b0 ;
    }
    ScanMux scanM SelectedBy dselA {
        1'b0 : regA[0] ;
        1'b1 : regB[0] ;
    }
    ClockMux clockM SelectedBy dselA {
        1'b0 : clkA ;
        1'b1 : clkB ;
    }
    DataMux dataM[3:0] SelectedBy dselB[2:0] {
        3'bx00 : regA[7:4] ;
        3'bx01 : regA[3:0] ;
        3'bx1x : regB[5:2] ;
    }
}
```

Related Topics

[ScanMux](#)

[ClockMux](#)

ClockMux

Creates and names a clock multiplexer.

Usage

```
Module module_name {
    ClockMux mux_name SelectedBy mux_select_signal {
        mux_select_value : clock_source_name ;
    }
}
```

Description

The ClockMux statement creates and names a clock multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. No signal types may be mixed for any of the multiplexers. The select signal of the multiplexers may be sourced only by data signals.

Arguments

- *mux_name*

A string that identifies the name of a data multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: [ScanRegister](#), [DataRegister](#), [ScanMux](#), [DataMux](#), [OneHotDataGroup](#), [OneHotScanGroup](#), [LogicSignal](#), [Alias](#).

Using this name, the outgoing signal of the multiplexer can be referenced. The width of *mux_name* must match the width of the *clock_source_name*.

- *mux_select_value : clock_source_name*

A repeatable string-string pair, connected by a colon symbol ‘:’. A DataMux must have at least one *mux_select_value - clock_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *clock_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Examples

The following shows usages of all the multiplexer types.

```
Module chip2 {
    ScanInPort siA ;
    ScanInPort siB ;
    ScanOutPort so      { Source scanM ; }
    ShiftEnPort sel ;
    CaptureEnPort ce1 ;
    UpdateEnPort ue1 ;
    ShiftEnPort se2 ;
    CaptureEnPort ce2 ;
    UpdateEnPort ue2 ;
    TCKPort tck ;
    ClockPort clkA ;
    ClockPort clkB ;
    ToClockPort clk { Source clockM ; }
    DataInPort dselA ;
    DataInPort dselB[2:0] ;
    DataOutPort do[3:0] { Source dataM[3:0] ; }
    ScanInterface cA12 {
        Port sel; Port ce1; Port ue1;
        Chain c1 { Port siA; Port so ; }
    }
    ScanInterface cB12 {
        Port se2; Port ce2; Port ue2;
        Chain c1 { Port siB; Port so ; }
    }
    ScanRegister regA[7:0] {
        ScanInSource siA ;
        ResetValue 8'b0 ;
    }
    ScanRegister regB[7:0] {
        ScanInSource siB ;
        ResetValue 8'b0 ;
    }
    ScanMux scanM SelectedBy dselA {
        1'b0 : regA[0] ;
        1'b1 : regB[0] ;
    }
    ClockMux clockM SelectedBy dselA {
        1'b0 : clkA ;
        1'b1 : clkB ;
    }
    DataMux dataM[3:0] SelectedBy dselB[2:0] {
        3'bx00 : regA[7:4] ;
        3'bx01 : regA[3:0] ;
        3'bx1x : regB[5:2] ;
    }
}
```

Related Topics

[ScanMux](#)

[DataMux](#)

OneHotDataGroup

Implements a convenient way of declaring a named, hierarchically distributed data multiplexer.

Usage

```
Module module_name {
    OneHotDataGroup onehotdatagroup_name {
        Instance ;
        DataRegister ;
        Port concat_signal ;
    }
}
```

Description

The OneHotDataGroup statement implements a convenient way of declaring a named, hierarchically distributed data multiplexer. You would use a OneHotDataGroup if various inputs to the multiplexer are sourced by data registers distributed across multiple instances. This statement simplifies the declaration of such a data multiplexer by automatically making the correct data input and data output connections, including read, write, and address connections.

A OneHotDataGroup may use any combination of the three options declaring the source input signal of the multiplexer. The order of the declarations is not relevant, since each source should have an enable signal defined in such a way that at most one source is selected at a time. The OneHotDataGroup then assumes the value of this selected, unique source, or the x-value, if none of the sources is selected, or if multiple data sources are selected simultaneously.

Arguments

- *onehotdatagroup_name*

A required string, declaring the name of the OneHotDataGroup. This name can be considered as the name of the distributed multiplexer. As usual, *onehotdatagroup_name* can be used to refer to the output signal of this multiplexer, for example as a source of a **DataOutPort** statement. The width of *onehotdatagroup_name* must be larger or equal to the width of the connected output ports of the referenced **Instance** or **DataRegister**.
onehotdatagroup_name assumes the value of the one selected source, or the x-value, if none of its sources are enabled.

- Instance

An optional, repeatable **Instance** statement. This instance must use the **AddressValue** parameter.

- DataRegister

An optional, repeatable **DataRegister** statement. This **DataRegister** must use the **AddressValue** parameter.

- *Port concat_signal*

An optional, repeatable keyword-string pair. *concat_signal* may reference a data output single signal or a concatenation of such signals. *concat_signal* may also be a hierarchical instance pathname ending at a data source signal. The referenced signal will be considered an input port of the multiplexer represented by the OneHotDataGroup statement. For the enable signal required for each source of the OneHotDataGroup's source signal, one of the following rules will apply:

- For a referenced DataOutPort, the Enable statement is included in the DataOutPort declaration.
- For a referenced DataOutPort, the Source parameter references another data signal, which has an enabling condition.
- The instance referenced in the instance pathname in a hierarchical *concat_signal* has an AddressValue parameter.
- A primary input port of the top level module is assumed to have an enabling condition from the outside.

Examples

The following shows an example of a OneHotDataGroup. Observe how the address of a **DataRegister** is determined. For example the **DataRegister** with the hierarchical name phy1.bank1.D1 has the address of $8'b01\ 000000 + 6'b0001\ 00 + 2'b01 = 8'b01\ 0001\ 01$. Similarly, the **DataRegister** phy2.R has the address $8'b10\ 000000 + 6'b0010\ 00 = 8'b10\ 0010\ 00$.

The Example 2 of the **Instance** statement shows a variation of this example.

```

Module top {
    AddressPort address[7:0] ;
    DataInPort dataIn[15:0] ;
    WriteEnPort write_enable ;
    ReadEnPort read_enable ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ; }
    OneHotDataGroup outMux[15:0] {
        Instance phy1 Of phy {
            AddressValue 2'b01, 6'b000000 ;
        }
        Instance phy2 Of phy {
            AddressValue 2'b10, 6'b000000 ;
        }
    }
}
Module phy {
    DataInPort dataIn1[7:0] ;
    DataInPort dataIn2[7:0] ;
    WriteEnPort we ;
    ReadEnPort re ;
    AddressPort address1[3:0] ;
    AddressPort address2[3:0] ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ; }
    OneHotDataGroup outMux[15:0] {
        DataRegister R[15:0] {
            AddressValue 4'b0010,2'b00 ;
        }
        Instance bank1 Of bank {
            AddressValue 4'b0001,2'b00 ;
        }
    }
}
Module bank {
    DataInPort dataIn[15:0] ;
    WriteEnPort we ;
    ReadEnPort re ;
    AddressPort address[1:0] ;
    DataOutPort dataOut[15:0] { Source outMux[15:0] ; }
    OneHotDataGroup outMux[15:0] {
        DataRegister D0[15:0] { AddressValue 2'b00 ; }
        DataRegister D1[15:0] { AddressValue 2'b01 ; }
        DataRegister D2[15:0] { AddressValue 2'b10 ; }
        DataRegister D3[15:0] { AddressValue 2'b11 ; }
    }
}

```

```
set_context patterns -ijtag
read_icl ./example_fixed.icl
set_current_design
set_system_mode analysis
open_pattern_set p1
iWrite phy1.bank1.D3[15:0] 0xaaaa
iApply
iRead phy2.R[15:0] 0xbbbb
iApply
close_pattern_set
write_pat p1.pdl -pdl -replace
cat p1.pdl
```

Related Topics

[Instance](#)

[OneHotScanGroup](#)

[DataRegister](#)

OneHotScanGroup

Implements a convenient way of declaring a named, hierarchically distributed scan multiplexer.

Usage

```
Module module_name {
    OneHotScanGroup onehotscangroup_name {
        Port source_name ;
    }
}
```

Description

The OneHotScanGroup statement implements a convenient way of declaring a named, hierarchically distributed scan multiplexer. You would use a OneHotScanGroup if various inputs to the multiplexer are sourced by scan-out ports distributed across multiple instances. This statement simplifies the declaration of such a scan multiplexer by automatically making the correct scan input and scan output connections.

Arguments

- *onehotscangroup_name*

A required string, declaring the name of the OneHotScanGroup. This name can be considered as the name of the distributed multiplexer. As usual, *onehotscangroup_name* can be used to refer to the output signal of this multiplexer, for example as a source of a [ScanOutPort](#) statement. The width of *source_name* must be equal to the width of *onehotdatagroup_name*. Each scan source must have an enable signal defined in such a way that at most one source is selected at a time when the OneHotScanGroup is in the active scan path. This enable signal is not necessarily directly at the ScanOutPort of *source_name*, but might be located deeper in the hierarchy, for example, through cascaded OneHotScanGroups. When selected, the *onehotdatagroup_name* then assumes the value of this selected source, or the x-value, if none of the sources is selected, or multiple sources would have been selected.

- *Port source_name*

A required, repeatable keyword-string pair. *source_name* may reference a scan output port single signal or a concatenation of such signals. *source_name* may also be a hierarchical instance pathname ending at a scan output port signal. The referenced scan output port signals will be considered an input port of the multiplexer represented by the OneHotScanGroup statement.

Examples

The following shows an example of a OneHotScanGroup. Observe how easy it is to declare the hierarchically distributed scan multiplexer “outScanMux” and the usage of the OneHotScanGroup’s name in the Source statement of the top level scan-out port.

```

Module chip_onehotscangroup {
    ScanInPort      si;
    ScanOutPort     so      { Source outScanMux; }
    ShiftEnPort     se;
    SelectPort      en[3:0];
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    OneHotScanGroup outScanMux {
        Port tdr_0.so ;
        Port tdr_1.so ;
        Port tdr_2.so ;
        Port tdr_3.so ;
    }
    Instance tdr_0 Of tdr {
        InputPort si = si ;
        InputPort en = en[0];
    }
    Instance tdr_1 Of tdr {
        InputPort si = si ;
        InputPort en = en[1];
    }
    Instance tdr_2 Of tdr {
        InputPort si = si ;
        InputPort en = en[2];
    }
    Instance tdr_3 Of tdr {
        InputPort si = si ;
        InputPort en = en[3];
    }
}
Module tdr {   ScanInPort      si;
    ScanOutPort     so      { Source R[0]; Enable en; }
    ShiftEnPort     se;
    SelectPort      en;
    CaptureEnPort   ce;
    UpdateEnPort    ue;
    TCKPort         tck;
    ScanRegister R[7:0] {
        ScanInSource   si;
        ResetValue     8'b0 ;
    }
}

```

Related Topics

[Instance](#)

[OneHotDataGroup](#)

[ScanRegister](#)

AccessLink

Links an ICL network to an 1149.1 TAP controller that is described in a BSDL file.

Usage

```
Module module_name {
    AccessLink instance_name Of STD_1149_1_2001 {
        BSDLEntity entity_name;
        instruction_name {
            ScanInterface { scan_interface_name; ... } // required
            ActiveSignals { signal_name; ... } // optional
        }
    }
}
```

Description

A wrapper that can be used inside the top level ICL Module to link an ICL network to an 1149.1 TAP controller that is described in a BSDL file. The tool currently only supports the BSDL format that is part of IEEE1149.1-2001.

When reading an ICL file which contains an ICL module with an AccessLink Of STD_1149_1_2001, the [read_icl](#) command must be used with the [-bsdl_file](#) option to provide the path to the referenced BSDLEntity. An error is generated if the specified BSDLEntity is not found in the specified BSDL file.

Arguments

- *instance_name*

A string used to define the instance name of the AccessLink. The string must start with a letter and only contain letters, numbers and underscores. As shown below, the *instance_name* is used when referring to the defined *signal_name* in an ICL connection. When the AccessLink is read in the tool, a TAP ICL module is created from the BSDL and instantiated in the top level ICL module in place of the AccessLink using the same *instance_name*. This is also illustrated below.

- *BSDLEntity entity_name;*

A required keyword-string pair which defines the name of the BSDL entity that is referenced by the AccessLink wrapper. The file pointed to by the “[read_icl -bsdl_file](#)” option must contain an “Entity *entity_name* is” clause.

- **instruction_name**

A required string that references one of the instruction names defined in the INSTRUCTION_OPCODE attribute of the BSDL file. When the specified *instruction_name* is loaded in the INSTRUCTION register, the *scan_interface_names* listed in the ScanInterface wrapper are placed along the active DR scan path between the TAP_SCAN_IN and TAP_SCAN_OUT ports.

- *scan_interface_name*

A repeatable string that references the ScanInterface of ICL module instances. The format of the string is a leaf_instance_name followed by an optional “.scan_interface_name” string. The “.scan_interface_name” string is optional when the ICL module associated with the leaf_instance_name only has one ScanInterface. If the ICL module associated to the leaf_instance_name has more than one ScanInterface wrapper, the “.scan_interface_name” string is required to identify one of them.

The scan_interface_name are serially daisy chained with the first one closer to TDI and the last one closer to TDO.

- *signal_name*

A repeatable string that defines signal_names that are high when the instruction_name is loaded into the INSTRUCTION register. When a signal_name is not specified inside a given instruction_name wrapper, the signal is low when the instruction_name is loaded into the INSTRUCTION register. The signal_names are identified as “instance_name.signal_name” when referenced to make connections in the ICL. This is illustrated below.

Examples

Below is an example of a top level ICL module containing an AccessLink. It references a BSDL file which contains the element shown below. After the elaboration which is triggered by the `set_current_design` command, you can see that a TAP controller was extracted from the BSDL file and instantiated into the top level ICL module. The TAP ports were also extracted and added to the ICL module. The original AccessLink wrapper is commented out. The ClockPorts and their original connections are left unaffected.

Note

 If an ICL module with an AccessLink statement has been loaded into memory by means of `read_icl`, then the commands `write_icl` and `report_icl_modules` do not result in usable ICL of this module before it has been subject to `set_current_design`.

Following is the content of mydesign.icl with an AccessLink wrapper. Notice how the ActiveSignal long_setup is referenced as “bsdl_tap.long_setup” when used to connect the “selectWIR” input port of the “core” instance.

```

Module mydesign {
    ClockPort CLK;
    AccessLink bsdl_tap Of STD_1149_1_2001 {
        BSDLEntity acme4523;
        BP0_LONG_SETUP {
            ScanInterface {core.ijtag0;}
            ActiveSignals {long_setup;}
        }
        BP0_SHORT_SETUP {
            ScanInterface {core.ijtag0;}
        }
        BP1_LONG_SETUP {
            ScanInterface {core.ijtag1;}
            ActiveSignals {long_setup;}
        }
        BP1_SHORT_SETUP {
            ScanInterface {core.ijtag1;}
        }
    }
}

Instance core Of core {
    InputPort selectWIR = bsdl_tap.long_setup;
    InputPort clk = CLK;
}
}

```

Following is the section from the BSDL file “acme4523.bsd1” that is used to extract the TAP controller.

```

entity acme4523 is
    attribute TAP_SCAN_RESET of TRST : signal is true;
    attribute TAP_SCAN_IN of TDI : signal is true;
    attribute TAP_SCAN_MODE of TMS : signal is true;
    attribute TAP_SCAN_OUT of TDO : signal is true;
    attribute TAP_SCAN_CLOCK of TCK : signal is (1.0e+07, BOTH);

    attribute INSTRUCTION_OPCODE of acme4523: entity is
        "IDCODE      (1111111111111110), " &
        "BYPASS      (1111111111111111), " &
        "EXTEST      (11111111111101000), " &
        "SAMPLE      (11111111111111000), " &
        "PRELOAD     (11111111111111000), " &
        "CLAMP       (1111111111111101111) " &
        "BP0_LONG_SETUP (XXXXXXXX000111XXX010), " &
        "BP1_LONG_SETUP (XXXXXXXX001101XXX010), " &
        "BP2_LONG_SETUP (XXXXXXXX001001XXX010), " &
        "BP3_LONG_SETUP (XXXXXXXX000101XXX010), " &
        "BP4_LONG_SETUP (XXXXXXXX000001XXX010), " &
        "BP0_SHORT_SETUP (XXXXXXXX110110XXX010), " &
        "BP1_SHORT_SETUP (XXXXXXXX111100XXX010), " &
        "BP2_SHORT_SETUP (XXXXXXXX111000XXX010), " &
        "BP3_SHORT_SETUP (XXXXXXXX110100XXX010), " &
        "BP4_SHORT_SETUP (XXXXXXXX110000XXX010) " ;

end acme4523;

```

Below is a dofile that loads the ICL modules in memory and elaborates them. The modified top level ICL module as well as the created TAP ICL modules are then reported. Notice how the TAP ICL module is called <bsdl_entity_name>_bsdl_tap. Its leaf instance name reused the one that was used by the AccessLink.

```

> set_context patterns -ijtag
> read_icl mydesign.icl -bsdl_file acme4523.bsdl
> read_icl core.icl
> set_current_design
> report_icl_module -module mydesign
Module mydesign {
    ClockPort CLK;
    ScanInPort TDI;
    TMSPort TMS;
    TCKPort TCK;
    TRSTPort TRST;
    ScanOutPort TDO {
        Source bsdl_tap.tdo;
    }
/*
    AccessLink bsdl_tap Of STD_1149_1_2001 {
        BSDLEntity acme4523;
        BP0_LONG_SETUP {
            ScanInterface {core.ijtag0;}
            ActiveSignals {long_setup;}
        }
        BP0_SHORT_SETUP {
            ScanInterface {core.ijtag0;}
        }
        BP1_LONG_SETUP {
            ScanInterface {core.ijtag1;}
            ActiveSignals {long_setup;}
        }
        BP1_SHORT_SETUP {
            ScanInterface {core.ijtag1;}
        }
    }
}
ScanMux tesson_access_link_mux0 SelectedBy
    bsdl_tap.BP0_LONG_SETUP_select,
    bsdl_tap.BP0_SHORT_SETUP_select,
    bsdl_tap.BP1_LONG_SETUP_select,
    bsdl_tap.BP1_SHORT_SETUP_select {
    4'b1xxx | 4'bx1xx | 4'bxx1x | 4'bxxx1 : TDI;
}
Instance core Of core {
    InputPort selectWIR = bsdl_tap.long_setup;
    InputPort clk = CLK;
    InputPort si = tesson_access_link_mux0;
}
Instance bsdl_tap Of acme4523_bsdl_tap {
    InputPort tdi = TDI;
    InputPort tms = TMS;
    InputPort tck = TCK;
    InputPort trst = TRST;
    InputPort BP0_LONG_SETUP_from_scan_out = core.fso0;
    InputPort BP0_SHORT_SETUP_from_scan_out = core.fso0;
    InputPort BP1_LONG_SETUP_from_scan_out = core.fso1;
    InputPort BP1_SHORT_SETUP_from_scan_out = core.fso1;
}
}

```

```
> report_icl_module -module acme4523_bsdl_tap
Module acme4523_bsdl_tap {
    TCKPort tck;
    TMSPort tms;
    TRSTPort trst;
    ScanInPort tdi;
    ScanOutPort tdo {
        Source IRMux;
    }
    ToResetPort testLogicReset {
        ActivePolarity 1;
        Source fsm.tlr;
    }
    ToResetPort testLogicResetInv {
        ActivePolarity 0;
        Source ~fsm.tlr;
    }
    ToCaptureEnPort BP0_LONG_SETUP_capture_en;
    ToShiftEnPort BP0_LONG_SETUP_shift_en;
    ToUpdateEnPort BP0_LONG_SETUP_update_en;
    ToSelectPort BP0_LONG_SETUP_select {
        Source BP0_LONG_SETUP_logic_signal;
    }
    ToResetPort BP0_LONG_SETUP_reset_inv {
        ActivePolarity 0;
    }
    ScanInPort BP0_LONG_SETUP_from_scan_out;
    ScanOutPort BP0_LONG_SETUP_to_scan_in {
        Source tdi;
    }
    ToCaptureEnPort BP0_SHORT_SETUP_capture_en;
    ToShiftEnPort BP0_SHORT_SETUP_shift_en;
    ToUpdateEnPort BP0_SHORT_SETUP_update_en;
    ToSelectPort BP0_SHORT_SETUP_select {
        Source BP0_SHORT_SETUP_logic_signal;
    }
    ToResetPort BP0_SHORT_SETUP_reset_inv {
        ActivePolarity 0;
    }
    ScanInPort BP0_SHORT_SETUP_from_scan_out;
    ScanOutPort BP0_SHORT_SETUP_to_scan_in {
        Source tdi;
    }
    ToCaptureEnPort BP1_LONG_SETUP_capture_en;
    ToShiftEnPort BP1_LONG_SETUP_shift_en;
    ToUpdateEnPort BP1_LONG_SETUP_update_en;
    ToSelectPort BP1_LONG_SETUP_select {
        Source BP1_LONG_SETUP_logic_signal;
    }
    ToResetPort BP1_LONG_SETUP_reset_inv {
        ActivePolarity 0;
    }
    ScanInPort BP1_LONG_SETUP_from_scan_out;
    ScanOutPort BP1_LONG_SETUP_to_scan_in {
        Source tdi;
    }
    ToCaptureEnPort BP1_SHORT_SETUP_capture_en;
    ToShiftEnPort BP1_SHORT_SETUP_shift_en;
```

```

ToUpdateEnPort BP1_SHORT_SETUP_update_en;
ToSelectPort BP1_SHORT_SETUP_select {
    SourceBP1_SHORT_SETUP_logic_signal;
}
ToResetPort BP1_SHORT_SETUP_reset_inv {
    ActivePolarity 0;
}
ScanInPort BP1_SHORT_SETUP_from_scan_out;
ScanOutPort BP1_SHORT_SETUP_to_scan_in{
    Source tdi;
}
DataOutPort long_setup{
    Source long_setup_LogicSignal;
}
ScanInterface Client {
    Port tdi;
    Port tdo;
    Port trst;
    Port tms;
    Port tck;
}
ScanInterface BP0_LONG_SETUP {
    Port BP0_LONG_SETUP_capture_en;
    Port BP0_LONG_SETUP_shift_en;
    Port BP0_LONG_SETUP_update_en;
    Port BP0_LONG_SETUP_select;
    Port BP0_LONG_SETUP_from_scan_out;
    Port BP0_LONG_SETUP_to_scan_in;
    Port BP0_LONG_SETUP_reset_inv;
}
ScanInterface BP0_SHORT_SETUP {
    Port BP0_SHORT_SETUP_capture_en;
    Port BP0_SHORT_SETUP_shift_en;
    Port BP0_SHORT_SETUP_update_en;
    Port BP0_SHORT_SETUP_select;
    Port BP0_SHORT_SETUP_from_scan_out;
    Port BP0_SHORT_SETUP_to_scan_in;
    Port BP0_SHORT_SETUP_reset_inv;
}
ScanInterface BP1_LONG_SETUP {
    Port BP1_LONG_SETUP_capture_en;
    Port BP1_LONG_SETUP_shift_en;
    Port BP1_LONG_SETUP_update_en;
    Port BP1_LONG_SETUP_select;
    Port BP1_LONG_SETUP_from_scan_out;
    Port BP1_LONG_SETUP_to_scan_in;
    Port BP1_LONG_SETUP_reset_inv;
}
ScanInterface BP1_SHORT_SETUP {
    Port BP1_SHORT_SETUP_capture_en;
    Port BP1_SHORT_SETUP_shift_en;
    Port BP1_SHORT_SETUP_update_en;
    Port BP1_SHORT_SETUP_select;
    Port BP1_SHORT_SETUP_from_scan_out;
    Port BP1_SHORT_SETUP_to_scan_in;
    Port BP1_SHORT_SETUP_reset_inv;
}
Attributetessent_instruction_reg = "INSTRUCTION";

```

```
Attribute tesson_tessent_bypass_reg = "BYPASS";
Attribute tesson_tessent_boundary_scan_reg = "BOUNDARY";
Attribute tesson_tessent_device_id_reg = "DEVICE_ID";
Attribute tesson_tessent_instrument_type = "mentor::jtag_bscan";
Attribute tesson_tessent_instrument_subtype = "tap";
Attribute tesson_tessent_signature = "a693927b83d713ce7521168994205e56";
Enum INSTRUCTION_OPCODES {
    BP2_SHORT_SETUP = 18'bxxxxxxxx111000xxx010;
    BP3_SHORT_SETUP = 18'bxxxxxxxx110100xxx010;
    BP0_SHORT_SETUP = 18'bxxxxxxxx110110xxx010;
    BP1_SHORT_SETUP = 18'bxxxxxxxx111100xxx010;
    BP4_SHORT_SETUP = 18'bxxxxxxxx110000xxx010;
    PRELOAD = 18'b111111111111111000;
    BYPASS = 18'b1111111111111111;
    IDCODE = 18'b1111111111111110;
    SAMPLE = 18'b1111111111111100;
    CLAMP = 18'b111111111111110111;
    EXTEST = 18'b1111111111111101000;
    BP4_LONG_SETUP = 18'bxxxxxxxx000001xxx010;
    BP2_LONG_SETUP = 18'bxxxxxxxx001001xxx010;
    BP3_LONG_SETUP = 18'bxxxxxxxx000101xxx010;
    BP0_LONG_SETUP = 18'bxxxxxxxx000111xxx010;
    BP1_LONG_SETUP = 18'bxxxxxxxx001101xxx010;
}
ScanRegister INSTRUCTION[17:0] {
    ScanInSource tdi;
    CaptureSource 18'bxxxxxxxxxxxxxxxxxxxx01;
    ResetValue 18'b11111111111111110;
    RefEnum INSTRUCTION_OPCODES;
}
ScanRegister BOUNDARY[41:0] {
    ScanInSource tdi;
}
ScanRegister BYPASS {
    ScanInSource tdi;
    CaptureSource 1'b0;
}
ScanRegister DEVICE_ID[31:0] {
    ScanInSource tdi;
    CaptureSource 32'b00010001001000110100001001000111;
}
ScanMux IRMux SelectedBy fsm.irSel {
    1'b1 : INSTRUCTION[0];
    1'b0 : DRMux;
}
ScanMux DRMux SelectedBy INSTRUCTION[17:0] {
    18'b1111111111111100 : BOUNDARY[0];
    18'b1111111111110100 : BOUNDARY[0];
    18'b11111111111100111 : BYPASS;
    18'b1111111111111111 : BYPASS;
    18'b111111111111110111 : BYPASS;
    18'b111111111111111110 : DEVICE_ID[0];
    18'bxxxxxxxx110110xxx010 : BP0_SHORT_SETUP_from_scan_out;
    18'bxxxxxxxx111100xxx010 : BP1_SHORT_SETUP_from_scan_out;
    18'bxxxxxxxx000111xxx010 : BP0_LONG_SETUP_from_scan_out;
    18'bxxxxxxxx001101xxx010 : BP1_LONG_SETUP_from_scan_out;
}
LogicSignal long_setup_LogicSignal {
```

```
(INSTRUCTION == BP0_LONG_SETUP) || (INSTRUCTION == BP1_LONG_SETUP);
}
LogicSignal BP0_LONG_SETUP_logic_signal {
    INSTRUCTION == BP0_LONG_SETUP;
}
LogicSignal BP0_SHORT_SETUP_logic_signal {
    INSTRUCTION == BP0_SHORT_SETUP;
}
LogicSignal BP1_LONG_SETUP_logic_signal {
    INSTRUCTION == BP1_LONG_SETUP;
}
LogicSignal BP1_SHORT_SETUP_logic_signal {
    INSTRUCTION == BP1_SHORT_SETUP;
}
Instance fsm Of TOP_bsdl_tap_fsm {
    InputPort tck = tck;
    InputPort tms = tms;
    InputPort trst = trst;
}
}
```

Related Topics

[read_icl](#)

NameSpace

Places all subsequent modules in a specific namespace.

Note

 Currently Tessent does not support the NameSpace statement.

Related Topics

[UseNameSpace](#)

UserNameSpace

References all subsequent modules from a specific namespace defined earlier through the NameSpace statement.

Note

 Currently Tessent does not support the UserNameSpace statement.

Related Topics

[NameSpace](#)

Chapter 14

Adding Custom Plugin Functionality

This chapter describes how you can augment the functionality of Tessent Shell with your own Tcl packages, commands and metadata definitions. It explains the concept of a plugin directory that you populate and how to get it auto loaded when Tessent Shell boots up.

The chapter also describes the best practice method to minimize the amount of code that is actually parsed at boot time and shows you how to have your code loaded on demand the first time your custom commands are used.

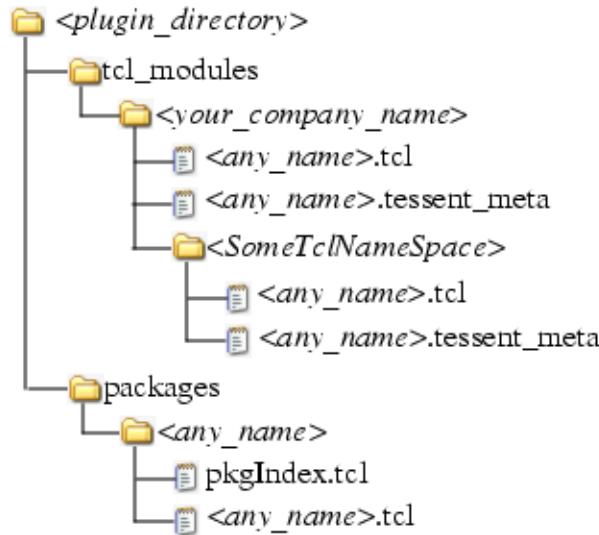
Finally, it describes how to create a proc called “u” to enable you to quickly reload the content of your plugin while you are developing and debugging it. The “u” proc is illustrated in [Example 1](#) for tcl_modules and in [Example 2](#) for packages.

plugin directory [3921](#)

plugin directory

A directory that contains special sub directories with tcl and metadata files to be auto loaded when Tessent Shell boots up.

Summary



Description

A directory that groups two subdirectories containing tcl code and metadata to load into Tesson Shell at boot time in order to augment its feature set with your custom Tcl code. The <plugin_directory> can be named anything you like. You point to your <plugin_directory> using the TESSENT_PLUGIN_PATH environment variable. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon.

You can also name your <plugin_directory> “tesson_plugin” and place it one level above the bin directory of Tesson Shell. By doing so, the directory will be automatically loaded without requiring the end user to set their TESSENT_PLUGIN_PATH environment variable. A user must define the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH to suppress the auto loading of this plugin directory.

The file *tesson_plugin/tcl_module/<your_company>/defaults.tcl* is the best place to include the definition of your company defaults as it will be automatically be seen by all users of the Tesson Shell tool. The Tcl file simply needs to include a this command:

```
read_config_data -from_string {
    DefaultsSpecification(company) {
        <your defaults here>
    }
}
```

The content and purpose of each sub directory and file is explained in the next section.

Components

- *tcl_modules/<your_company_name>*

The directory “tcl_modules” is where you define new commands and new metadata. The name “tcl_modules” cannot be changed. Inside this directory, you use an arbitrary sub-directory name which is typically the name of your company. If you are part of a large company with many divisions and you don’t want to have to coordinate with them the names of your tcl modules, you can also append your division name to your company name.

- <any_name>.tcl

All the tcl files specified at this level are loaded when Tesson Shell starts. You want to limit the amount of Tcl code those files contains so as not to slow down the Tesson Shell boot time. Those files should only include attribute and command registrations and the registration must be declared within a “namespace eval <SomeNameSpace>” block as shown in the [Example 1](#) below. In the same example, notice how the “create_wrapper_proc” command is used around the proc name of the “[register_tcl_command -tcl_proc](#)” argument to add special code that knows to load the body of the proc from files named <SomeTclNameSpace>/*.tcl when the command is called for the first time. The wrapper proc also disables tool command

transcribing while running your proc and restores it to its original setting after your command is done. You can always use the “[set_transcript_style full](#)” command in your procs if you want them to still transcript or enable transcripting on demand based on an option like it is done with the “[process_dft_specification - transcript_insertion_commands](#)” option. The wrapper proc provides handling of the Ctrl-C interrupt such that it terminates your command with the appropriate error message and error code. Finally, if your proc has a bug which results in a tcl command to fail, it will echo the detailed error message with the calling stack so that you can quickly identify the location of the problem. See [Example 1](#) below on how to create a proc called “u” to quickly reload the body of your procs after you have corrected issues.

- <any_name>.tessent_meta

All the files having an extension “.tessent_meta” are read in the tool at boot time. The files are sorted alphabetically before being read. See the section about the “[Metadata Configuration Syntax](#)” on page 3719 for an explanation of the syntax and when you may want to use it. Realize that the metadata loading order is important when you are using the “reference” property as documented in the [Wrapper](#) section because the referenced elements must have been read before they are referenced.

The alphabetical ordering of the files provides you the ability to control read ordering. Also note that metadata files contained in the <SomeTclNameSpace> sub-directories are always loaded after the one at the current level.

- <SomeTclNameSpace>/<any_name>.tcl

The directory name <SomeTclNameSpace> must match the namespace name used in the <any_name>.tcl files above it. The [create_wrapper_proc](#) command uses the leaf namespace from where it is called to know which files to auto load the first time it is run. For example, if the [create_wrapper_proc](#) command is invoked within the “[namespace eval ABC {}](#)” block, it will automatically source any file matching “./ABC/*.tcl”. The tcl code found inside the “ABC/*.tcl” do not need to be inside a “[namespace eval ABC {}](#)” block as the wrapper proc will automatically add them to the namespace.

- <SomeTclNameSpace>/<any_name>.tessent_meta

Those files are loaded at boot time. They are loaded after the file found one level above. See the discussion in the “<any_name>.tessent_meta” section to find out when that may be important. It is best to place your “.tessent_meta” file at this lower level to avoid cluttering the “[tcl_modules/<your_company_name>](#)” directory.

- packages

This directory contains sub directories which are automatically added to the Tcl autopath search when the “[package require](#)” command is invoked. Each sub-directory must each contains a file called [pkgIndex.tcl](#) and *.tcl files which get loaded when the given “[package require](#)” command is issued.

- <any_name>/pkgIndex.tcl

This file has a very specific format. The format is shown below and is used to load a series of Tcl files when the associated package is requested. It is recommended to define <my_package> and <my_namespace> identically and using this format ::<my_company>::<my_feature>. With this convention, you can easily request the package and import its exposed commands to your local namespace as shown in [Example 2](#). The <file_name> strings are the name of the Tcl file implementing features of the package and they must exist in the same directory.

```
package ifneeded <my_package> 1.0 \
    [string map [list @ $dir ! <my_namespace>] {
        no_transcript { read_tcl [file join {@} <file_name>.tcl] \
            -namespace {!} }
        no_transcript { read_tcl [file join {@} <file_name>.tcl] \
            -namespace {!} }
        package provide <my_package> 1.0
    }]
}
```

- <any_name>/<any_name>.tcl

Those Tcl files contain procs which are to be made available to the user when he requests the package. You include a “namespace export <proc_name>” on top of each proc you want to be exportable to the end user. Leave out the “namespace export” command for any internal proc not intended for the end user. See [Example 2](#) below for an example of such files.

Example 1

This example shows you how to create a plugin that defines a tcl_module implementing commands used to manage stacks.

Following the directory structure described above, these two files are created:

```
my_plugin/tcl_modules/my_company/stack_commands.tcl
my_plugin/tcl_modules/my_company/stack_commands/command_body.tcl
```

As you can see below, the file “stack_command.tcl” contains only register_tcl_command invocations inside a “namespace eval stack_commands” Tcl block as well as a variable definition. When any of those registered Tcl commands are called, it will call the wrapper proc that was created by the “create_wrapper_proc” command. This wrapper proc will load the file “stack_commands/command_body.tcl” the first time it is called and then call the proc it has just loaded.

The example uses a namespace variable called stack to hold an array of stacks. Each sub-command which operates on the stacks has access to common data through this variable.

To have this plugin automatically loaded into Tessent Shell, you simply need to set the environment variable TESSENT_PLUGIN_PATH to point to it. In this example, the variable points to my_plugin and his_plugin:

```
setenv TESSENT_PLUGIN_PATH ~/my_plugin:~him/his_plugin
```

In the example below, imagine the new_stack proc had a typo and the second to last line was using the undefined variable \$idd instead of \$id as it should. You would get an error when you tried to use it as shown in the transcript below. Notice how the error message provides the full context of the error including the calling stack. This information is essential for locating errors especially as the code becomes complex.

These kinds of mistakes are very common as you develop code and below is a useful method to fix them easily. Define a tcl proc called “u” which you can use to reload the *.tcl files found in the sub directory. That way, you can edit the source file, save the update, type the u command and test the updated files without having to restart Tessent Shell. An example definition of a “u” proc is shown below. Once the “u” command was run and it reloaded the corrected file, you can see the proper operation of the stack commands in the transcript below.

```
> proc u {} {
    namespace eval ::tcl_modules::my_company::stack_commands {
        foreach file_name \
            [glob ../my_plugin/tcl_modules/my_company/stack_commands/*.tcl] {
            puts "updating $file_name"
            source $file_name
        }
    }
}
set s [new_stack]
can't read "idd": no such variable
    while executing
"set stack($idd) [list]"
    (procedure "new_stack" line 8)
    invoked from within
"new_stack {*}$args"
=====
TCL calling stack
    ::tcl_modules::my_company::stack_commands::new_stack.wrapper --
namespace :::
=====
> u
updating ../my_plugin/tcl_modules/my_company/stack_commands/
command_body.tcl
> set s [new_stack]
> push $s v1
> push $s v2
> puts [pop $s]
v2
> puts [top_of $s]
v1
> puts [pop $s]
v1
> puts [pop $s]
```

Content of the stack_commands.tcl file:

```
namespace eval stack_commands {
    variable stack register_tcl_command push \
        -context_list {all} \
        -system_mode_list {setup analysis insertion} \
        -string_value {<stack_id> stack_id} \
        -string_value {<value> value} \
        -syntax_bnf {<stack_id> <value>} \
        -tcl_proc [create_wrapper_proc push]
    register_tcl_command pop \
        -context_list {all} \
        -system_mode_list {setup analysis insertion} \
        -string_value {<stack_id> stack_id} \
        -syntax_bnf {<stack_id>} \
        -tcl_proc [create_wrapper_proc pop]
    register_tcl_command top_of \
        -context_list {all} \
        -system_mode_list {setup analysis insertion} \
        -string_value {<stack_id> stack_id} \
        -syntax_bnf {<stack_id>} \
        -tcl_proc [create_wrapper_proc top_of]
    register_tcl_command new_stack \
        -context_list {all} \
        -system_mode_list {setup analysis insertion} \
        -syntax_bnf {} \
        -tcl_proc [create_wrapper_proc new_stack]
}
```

Content of the stack_commands/command_body.tcl file:

```
proc pop {args} {
    variable stack
    array set ARGS $args
    set stack_id $ARGS(stack_id)
    check_id $stack_id
    if {[llength $stack($stack_id)] > 0} {
        set res [lindex $stack($stack_id) end]
        set stack($stack_id) [lreplace $stack($stack_id) end end]
    } else {
        set res ""
    }
    return $res
}
proc push {args} {
    variable stack
    array set ARGS $args
    set stack_id $ARGS(stack_id)
    set value $ARGS(value)
    check_id $stack_id
    lappend stack($stack_id) $value
}
```

```
proc top_of {args} {
    variable stack
    array set ARGS $args
    set stack_id $ARGS(stack_id)
    check_id $stack_id
    if {[llength $stack($stack_id)] > 0} {
        set res [lindex $stack($stack_id) end]
    } else {
        set res ""
    }
    return $res
}
proc new_stack {args} {
    variable stack
    if {[array exists stack]} {
        set id [expr {[:tcl::mathfunc::max {*} [array names stack]] + 1}]
    } else {
        set id 0
    }
    set stack($id) [list]
    return $id
}
# local utility proc proc check_id {stack_id} {
variable stack
if {! [info exists stack($stack_id)]} {
    display_message "The stack id '$stack_id' is not active."
    return -code error
}
}
```

Example 2

This example shows you how to define a package which, when requested by the user, loads and exposes a series of debug utility procs. The files for this package are the following:

```
my_plugin/packages/debug_utilities/pkgIndex.tcl
my_plugin/packages/debug_utilities/timer_procs.tcl
my_plugin/packages/debug_utilities/conversion_procs.tcl
```

As you can see below, the user simply needs to issue the “package require” command to request the loading of the package. One can use the loaded command by specifying the complete namespace path to the commands such as

“`::my_company::debug_utilities::set_timer_start_point A`” but it is more convenient to import the commands into the local namespace using the “namespace import” command as shown below. After that, the procs becomes available locally as if they were defined in the local namespace.

Notice also the “u” proc in the example below. It is very useful again to enable debugging the source code of the procs without having to restart Tessian Shell. The “u” proc is slightly different than the one shown in [Example 1](#) as the pkgIndex.tcl file must be excluded from the reload.

```

> package require ::my_company::debug_utilities
1.0
> namespace import ::my_company::debug_utilities::*
> puts [join [info procs *timer*] \n]
report_timer_names
get_time_from_timer_start_point
set_timer_start_point
> puts [join [info procs *convert*] \n]
convert_to_wdhms
>
> proc u {} {
    namespace eval ::my_company::debug_utilities {
        set file_list [glob ../my_plugin/packages/debug_utilities/*.tcl]
        set pkgIndex_location [lsearch -glob $file_list */pkgIndex.tcl]
        foreach file_name [lreplace $file_list $pkgIndex_location \
                           $pkgIndex_location] {
            puts "updating $file_name"
            source $file_name
        }
    }
}
>
> set_timer_start_point A
Setting timer 'A' start time at '15:01:21'
> set_timer_start_point B
Setting timer 'B' start time at '15:01:21'
> get_time_from_timer_start_point A
weeks 0 days 0 hours 0 minutes 0 seconds 2
> get_time_from_timer_start_point B
weeks 0 days 0 hours 0 minutes 0 seconds 7

```

Content of the debug_utilities/pkgIndex.tcl file:

```

package ifneeded ::my_company::debug_utilities 1.0 \
    [string map [list @ $dir ! ::my_company::debug_utilities] {
        no_transcript { read_tcl [file join {@} timer_procs.tcl] \
                        -namespace {!} }
        no_transcript { read_tcl [file join {@} conversion_procs.tcl] \
                        -namespace {!} }
    }]
    package provide ::my_company::debug_utilities 1.0
}]
```

Content of the debug_utilities/timer_procs.tcl file:

```
variable timer_start_point_dict [dict create]
namespace export set_timer_start_point
proc set_timer_start_point {timer_name} {
    variable timer_start_point_dict
    dict set timer_start_point_dict $timer_name [clock seconds]
    puts "Setting timer '$timer_name' start time at '[clock format [clock
seconds] -format "%H:%M:%S"]'"
}
namespace export report_timer_names
proc report_timer_names {} {
    variable timer_start_point_dict
    if {[dict size $timer_start_point_dict] > 0} {
        set timer_start_points [dict keys $timer_start_point_dict]
        puts "Available timer names are:\n  [join $timer_start_points "\n  "]"
    } else {
        puts "No timer name exists"
    }
}
namespace export get_time_from_timer_start_point
proc get_time_from_timer_start_point {timer_name} {
    variable timer_start_point_dict
    if {[dict exist $timer_start_point_dict $timer_name]} {
        set start_point_time_val \
            [dict get $timer_start_point_dict $timer_name]
        set time_diff [expr [clock seconds] - $start_point_time_val]
        set converted_time_dict [convert_to_wdhms $time_diff]
        return $converted_time_dict
    } else {
        display_message -error "Timer name '$timer_name' does not exist"
        return -code error
    }
}
```

Content of the debug_utilities/conversion_procs.tcl file:

```
namespace export convert_to_wdhms
proc convert_to_wdhms {seconds} {
    if { ![string is digit -strict $seconds] } {
        display_message -error \
            "The supplied number is not an integer: '$seconds'"
        return -code error
    }
    set week_val      [expr $seconds/604800]
    set seconds      [expr $seconds - ($week_val *604800)]
    set day_val       [expr $seconds/86400]
    set seconds      [expr $seconds - ($day_val *86400)]
    set hour_val      [expr $seconds/3600]
    set seconds      [expr $seconds - ($hour_val*3600)]
    set minute_val   [expr $seconds/60]
    set second_val   [expr ($seconds - ($minute_val*60))]
    set time_conversion_results_dict \
        [dict create weeks $week_val \
                           days $day_val \
                           hours $hour_val \
                           minutes $minute_val \
                           seconds $second_val]
    return $time_conversion_results_dict
}
```

Related Topics

[register_tcl_command](#)

[register_attribute](#)

Chapter 15

Generation of ScanDEF

This chapter provides detailed information about ScanDEF.

ScanDEF File Generation	3931
ScanDEF File Format	3932

ScanDEF File Generation

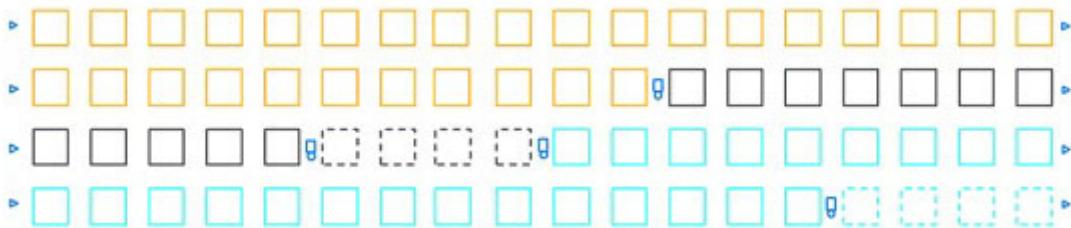
You use the command `write_scan_order` to generate the ScanDEF file.

In the legacy Tessent Shell scan insertion flow (which is equivalent to a single scan mode environment), every scan chain's make is unique and therefore can be written out into a scanDEF file using a simple structure. However, the same scan elements can be stitched into different scan chains in the Hierarchical Scan Insertion flow using a multi scan mode environment. Furthermore, since a single scanDEF file written out in the new flow would have to contain all scan elements regardless of the scan mode, it uses the unique scan segments rather than the unique scan chains to create the scan element groupings used by the layout tools for the scan chains optimization.

Each scan segment will comprise a set of scan path cells of a scan chain between the so called persistent components, which are buffer, mux, and/or (NOTE: only 2x1 actual MUXs are used; so, when there are more than 2 inputs, the AND/OR muxing is being used), and retiming cells. These persistent components will be written out into the START/STOP sections of each scanDEF segment corresponding to the scan segment, while the scan path cells of a scan segment will be written out into the ORDERED and/or FLOATING lists in the body section of a scanDEF segment.

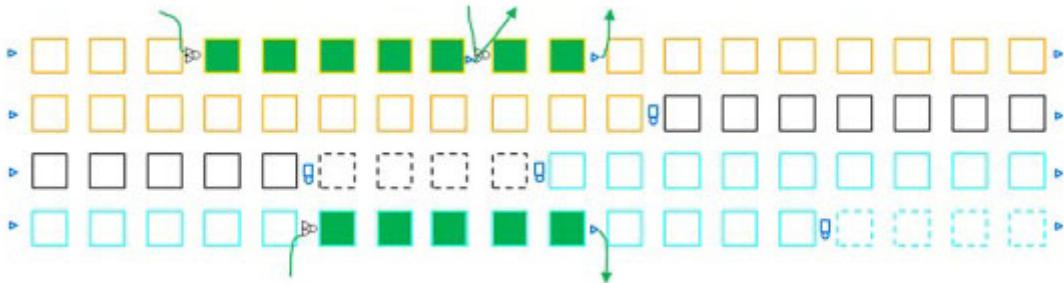
In the simplest case of scan chains in the single scan mode environment (see [Figure 15-1](#)), with four scan chains and three clock domains, each scan chain will be sub divided into the scan segments based on the clock/edge domain information, where the persistent components will consist of a buffer at the beginning and end of each scan chain and a retiming cell at the clock/edge domain transition points.

Figure 15-1. Single Scan Mode Environment



In the case of multi mode environment (see [Figure 15-2](#)), having InTest and ExTest scan modes, with four scan chains in the InTest mode, three wrapper chains in the ExTest mode, and three clock domains, each scan chain will be subdivided into the scan segments based on the clock/edge domain information plus the scan modes scan chains information. The persistent components in this case will consist of a buffer or mux (and/or) at the beginning of each scan chain, a buffer at the end of each scan chain, and a retiming cell at the clock/edge domain transition points.

Figure 15-2. Multi Scan Mode Environment



In general, the scan insertion tool always ensures that persistent gates exist at the beginning and end of scan chains, and when transitioning between constraint sets that need it. In addition, for power domain island transitions, persistent gates get inserted at the boundary of each power domain islands in order to simplify the insertion of level shifters.

Modules which have the “`exclude_from_concatenated_netlist`” attribute specified are considered as hard macros and are not included in the concatenated netlist. Moreover, when writing the scanDEF file, any traced scan elements from existing segments on these modules are omitted between the `scanIn` and `scanOut` sections of these segments, and stop at the SI/SO boundary, as is done for a cell library.

ScanDEF File Format

The new scanDEF file will contain all unique scan segments contributing to all scan chains in all scan modes.

The following is an example of a typical scanDEF segment:

```
- scan_segment_0
+ START tessent_persistent_cell_mux_stm1s1_intsi17_intsi13_i Y
+ ORDERED
    uWA/uA/f1 ( IN SI ) ( OUT Q )
    uWA/uA/bb/f2 ( IN SI ) ( OUT so )
    uWA/uA/f3 ( IN SI ) ( OUT Q )
+ FLOATING
    uWA/uA/f21 ( IN SI ) ( OUT Q )
+ STOP uWA/tessent_persistent_cell_buf_extso6_i A
# Partition constraints - clock domain: clk(+); chain type: core; scan
mode(s): InTest ExTest
+ PARTITION partition_1 MAXBITS 4 ;
```

Each scanDEF segment will start with unique segment name with following format:

scan_segment_%d

The START section will reference an output pin of a persistent component, which will be either a buffer, mux, or a retiming cell.

The STOP section will reference an input pin of a persistent component, which will be either a buffer or a retiming cell.

The body of a scanDEF segment will typically contain one or more FLOATING and/or ORDERED lists. Currently, an ORDERED list will be created for the cells on the known sub chain scan path (unless sub chain contains only 1 element) and the cells in a shift register. All other cells will be written out into the FLOATING lists.

In addition, a scanDEF segment will contain a PARTITION name. The PARTITION statement allows reordering tools to determine inter-segment compatibility for element swapping (both FLOATING elements and ORDERED elements): i.e. partition names determine the segments compatibility for repartitioning by swapping elements between them. Thus, segments with matching PARTITION names constitute a swap-compatible group. The length of the segments included in the same partition is specified by the MAXBITS constraint, which will be typically set to the number of elements in the segment's body. NOTE: if a PARTITION name is not provided, the segment is assumed to be in its own single partition and reordering can be performed only within this segment.

PARTITION name is generated based on the following criteria for a given scan segment: clock/edge domain, clusters this segment's scan chains belong to, power domains this segment's scan chains belong to, chain type (core, input_wrapper, output_wrapper), scan modes this segment's scan chains belong to. The mentioned criteria will be listed in the "Partition constraints" comment line.

scanDEF File Example

For the following scan chains configuration:
two scan modes, four scan chains in InTest, four scan chains in ExTest, four sub chains.

```
// command: report_scan_cells -scan_mode InTest
Scan mode 'InTest' scan cells:
-----
CellNo ChainName      GroupName Pathname   CellName ScanOut Clock ClockPolarity
-----
0  InTest_chain1 (schc1) grp1  /uWA/uA/f21 sff  Q    clk  (+)
1  InTest_chain1 (schi1) grp1  /uWA/uA/f1  sff  Q    clk  (+)
2  InTest_chain1 (schi1) grp1  /uWA/uA/bb/f2 sff  Q    clk  (+)
3  InTest_chain1 (schi1) grp1  /uWA/uA/f3  sff  Q    clk  (+)
0  InTest_chain2  grp1  /f1    sff  Q    clk2 (+)
0  InTest_chain3 (scho1) grp1  /uWA/uA/f31 sff  Q    clk  (+)
0  InTest_chain4 (schi2) grp1  /uB/f1    sff  Q    clk2 (+)
1  InTest_chain4 (schi2) grp1  /uB/f2    sff  Q    clk2 (+)
2  InTest_chain4 (schi2) grp1  /uB/f3    sff  Q    clk2 (+)
-----
// command: report scan cells -scan_mode ExTest
Scan mode 'ExTest' scan cells:
-----
CellNo ChainName      GroupName Pathname   CellName ScanOut Clock ClockPolarity
-----
0  ExTest_chain1 (schc1) grp1  /uWA/uA/f21 sff  Q    clk  (+)
1  ExTest_chain1 (schi1) grp1  /uWA/uA/f1  sff  Q    clk  (+)
2  ExTest_chain1 (schi1) grp1  /uWA/uA/bb/f2 sff  Q    clk  (+)
3  ExTest_chain1 (schi1) grp1  /uWA/uA/f3  sff  Q    clk  (+)
0  ExTest_chain2  grp1  /f1    sff  Q    clk2 (+)
0  ExTest_chain3 (scho1) grp1  /uWA/uA/f31 sff  Q    clk  (+)
0  ExTest_chain4 (schi2) grp1  /uB/f1    sff  Q    clk2 (+)
1  ExTest_chain4 (schi2) grp1  /uB/f2    sff  Q    clk2 (+)
2  ExTest_chain4 (schi2) grp1  /uB/f3    sff  Q    clk2 (+)
```

The scanDEF output will look as follows:

```
#  
VERSION 5.7 ;  
DIVIDERCHAR "/" ;  
BUSBITCHARS "[]" ;  
DESIGN A ;  
UNITS DISTANCE MICRONS 1000 ;  
  
SCANCHAINS 2 ;  
  
- chain1_sub0  
+ START ud8 QB  
+ ORDERED  
    ud9 ( IN SI ) ( OUT Q )  
    ud10 ( IN D ) ( OUT Q )  
+ ORDERED  
    ud1 ( IN SI ) ( OUT Q )  
    ud2 ( IN D ) ( OUT Q )  
    ud3 ( IN D ) ( OUT Q )  
    ud4 ( IN D ) ( OUT QB )  
+ STOP ud5 SI  
  
# Partition for core chain in clock clk1 (pos-edge) domain  
+ PARTITION partition_1 MAXBITS 6 ;  
  
- chain2_sub0  
+ START ud15 Q  
+ FLOATING  
    ud14 ( IN SI ) ( OUT Q )  
    ud13 ( IN SI ) ( OUT Q )  
    ud12 ( IN SI ) ( OUT Q )  
+ ORDERED  
    ud6 ( IN SI ) ( OUT Q )  
    ud7 ( IN D ) ( OUT QB )  
+ STOP ud11 SI  
  
# Partition for core chain in clock clk1 (pos-edge) domain  
+ PARTITION partition_1 MAXBITS 5 ;  
  
END SCANCHAINS  
  
END DESIGN
```


Appendix A

Parameter File Format and Keywords

This appendix contains reference material about the parameter file statements.

What is a Parameter File?.....	3940
Parameter File Syntax.....	3940
Parameter File Keywords	3942
A Note about the ALL_* Parameter File Keywords	3945
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS.....	3945
ALL_EXCLUDE_POWER_GROUND.....	3946
ALL_EXCLUDE_UNUSED	3946
ALL_FIXED_CYCLES	3947
ALL_FLATTEN_TIMING	3948
ALL_FULL_TIMEPLATES	3948
ALL_HEADER_COMMENT	3948
ALL_IDDQ_TESTER_CYCLE.....	3949
ALL_MAX_LOOP	3950
ALL_MIN_SCAN_LOAD.....	3951
ALL_NO_CYCLE_OPT	3951
ALL_NO_LOOP	3951
ALL_NO_PATTERN_TYPE.....	3951
ALL_NONSCAN_CONSTANT	3952
ALL_ORIGINAL_INDEX.....	3952
ALL_PRESERVE_FORCE_X.....	3953
ALL_TIME_RESOLUTION	3953
ALL_USE_CYCLE_LOOP	3953
CTL_ONE_PAT	3954
CTL_NO_PROTO	3954
CTL_STIL_0	3954
CTL_STIL_1	3955
FTDL_DESIGNER.....	3955
FTDL_MAX_PAT_BLOCK	3955
FTDL_REVISION	3956
FTDL_TCOMMENT	3956
FTDL_TEST_NAME	3956
FTDL_TEST_NUM	3957
FTDL_TEST_SUFFIX.....	3957
FTDL_ZMODE_MES	3957
SIM_ANNOTATE QUIET	3958
SIM_CHAIN_ERROR	3958
SIM_CHANGE_PATH	3959

SIM_COMPARE_SUMMARY	3959
SIM_COMPARE_X	3959
SIM_DELAY_SCAN_RELEASE	3960
SIM_DIAG_FILE	3961
SIM_DISTRIBUTED_PAT	3961
SIM_DUMPFILE_PATH	3962
SIM_EARLY_RELEASE	3962
SIM_EARLY_RELEASE_LAST	3963
SIM_EARLY_RELEASE_TIME	3963
SIM_FILL_BIDISCAN	3964
SIM_FORCE_Z_AT_CYCLE	3964
SIM_INCLUDE	3964
SIM_INSTANCE_NAME	3965
SIM_KEEP_PATH	3965
SIM_MASK_FILE	3966
SIM_MIN_NAME_FILE	3966
SIM_MISCOMPARE_LIMIT	3967
SIM_MODULE_INCLUDE	3967
SIM_MODULE_NAME	3967
SIM_NAME_FILE	3968
SIM_NO_CHAINNAME_FILE	3968
SIM_NO_INTERNAL_COMPARE	3969
SIM_NO_MEASURE_IN	3969
SIM_PARALLEL_DELAY_SHIFT_CLOCKS	3969
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE	3970
SIM_POST_SHIFT	3970
SIM_PRE_SHIFT	3971
SIM_PRECISION	3971
SIM_SERIAL_CHAIN	3971
SIM_SHIFT_DEBUG	3972
SIM_SIGNAL_SPY	3973
SIM_STATUS_MSG	3973
SIM_STRETCH_SCAN_RELEASE	3973
SIM_TIMEPLATE_COMM	3974
SIM_TMP_REG_LENGTH	3974
SIM_TOP_NAME	3974
SIM_VECTOR_COMM	3975
SIM_VECTYPE_SIGNAL	3976
STIL_2005_SCAN	3977
STIL_CHAIN_CELLS	3978
STIL_CHANNEL_CELLS	3978
STIL_CLOCK_WFC_PULSE	3979
STIL_DUAL_MODE	3979
STIL_FULL_CHAIN	3980
STIL_IDDQ_UPPERCASE	3980

STIL_MIN_QUOTE	3980
STIL_NESTED_MACRO	3981
STIL_NO_SCANSTRUCTURE	3981
STIL_NO_SCANX	3981
STIL_NOMEASURE_CLOCK	3982
STIL_PAT_ANN	3982
STIL_PAT_CMT	3983
STIL_PAT_LAB	3983
STIL_QUAD_MODE	3983
STIL_QUOTE_ALL	3984
STIL_SCAN_ANN1	3984
STIL_STRUCTURAL	3985
STIL_TI_TITLE	3987
STIL_TRIM_PULSE	3988
STIL_VECTOR_ANN	3988
STIL_VERG_ESC	3990
STIL_WRAP_ANNOTATION	3990
TITDL_CHAIN_SETTYPE	3990
TITDL_CUSTOMER	3991
TITDL_GROUP_TIMING	3991
TITDL_KEEP_SCANOUT	3991
TITDL_LIBRARY	3992
TITDL_PARTNUM	3992
TITDL_PSEUDO_PREFIX	3992
TITDL_REVISION	3993
TITDL_SET_DESCRIPTION	3994
TITDL_SETNAME	3994
TITDL_SETTYPE	3994
WGL_ADD_LASTX	3995
WGL_ALT_BIDI	3995
WGL_ALT_VECT_ANN	3996
WGL_EDGE_STROBE	3996
WGL_FULL_CHAIN	3997
WGL_FULL_SCANGROUP	3998
WGL_GROUP_PIN	3999
WGL_INV_SC	3999
WGL_NO_SCANX	4000
WGL_NOMEASURE_CLOCK	4000
WGL_ONE_ILLINOIS	4001
WGL_PATTERN_NAME	4001
WGL_TRIM.LEADING_P	4001
WGL_TRIM_PULSE	4002
WGL_VECTOR_ANN	4002
WGL_VERG_ESC	4003
WGL_VTRAN_PADSC	4003

WGL_WRAP_ANNOTATION	4003
-------------------------------	------

What is a Parameter File?

The parameter file specifies field names and values for use in certain Vector Interfaces output formats.

The tool uses this file when you specify with the “[write_patterns](#) -Parameter” command.

The tool adds a section at the beginning of STIL, WGL, and Verilog output files that captures the environment data and the switches specified with the `write_patterns` command.

- In the STIL format, this information is added to the HEADER {} block as an annotation.
- In the WGL format, this information is added to the header block as a series of annotation statements.
- In the Verilog format, this information is added to the beginning of the Verilog test bench as a comment.

Parameter File Syntax

The parameter file is a text file containing a series of parameter statements.

Each statement uses the following format:

```
KEYWORD arguments;
```

Where:

- **UPPERCASE** — Indicates a keyword and must be entered exactly as shown.
- ***Italic*** — Indicates a user-supplied argument.
- {} — Indicates a multi-option argument. Include only one option and do not enter the braces.
- **Underline** — Indicates the default setting for a multi-option argument.

The parameter file has the following rules:

- All parameter statements must end with a semicolon.
- Comments are marked with double slashes “//” at the beginning of a line. The tool ignores everything between the slashes and the end of line character.
- You can use each parameter only once per file. If you use a parameter more than once, the latter entry overrides the previous one.

- Each parameter statement must occupy a single line. You cannot embed carriage returns within a parameter statement.
- Each parameter statement line cannot exceed 2048 characters. There are no restrictions on the parameter file size.

Note

 All parameters are optional, so you can include or exclude as needed. When you omit a parameter, the tool uses the default setting. Refer to each parameter description for the default behavior.

Parameter File Keywords

This is a list of the available parameter statements.

A Note about the ALL_* Parameter File Keywords	3945
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS	3945
ALL_EXCLUDE_POWER_GROUND.....	3946
ALL_EXCLUDE_UNUSED	3946
ALL_FIXED_CYCLES.....	3947
ALL_FLATTEN_TIMING.....	3948
ALL_FULL_TIMEPLATES	3948
ALL_HEADER_COMMENT	3948
ALL_IDDQ_TESTER_CYCLE.....	3949
ALL_MAX_LOOP	3950
ALL_MIN_SCAN_LOAD.....	3951
ALL_NO_CYCLE_OPT.....	3951
ALL_NO_LOOP	3951
ALL_NO_PATTERN_TYPE	3951
ALL_NONSCAN_CONSTANT.....	3952
ALL_ORIGINAL_INDEX	3952
ALL_PRESERVE_FORCE_X.....	3953
ALL_TIME_RESOLUTION	3953
ALL_USE_CYCLE_LOOP	3953
CTL_ONE_PAT.....	3954
CTL_NO_PROTO.....	3954
CTL_STIL_0	3954
CTL_STIL_1	3955
FTDL_DESIGNER	3955
FTDL_MAX_PAT_BLOCK.....	3955
FTDL_REVISION.....	3956
FTDL_TCOMMENT	3956
FTDL_TEST_NAME	3956
FTDL_TEST_NUM.....	3957
FTDL_TEST_SUFFIX	3957
FTDL_ZMODE_MES.....	3957
SIM_ANNOTATE QUIET	3958

SIM_CHAIN_ERROR	3958
SIM_CHANGE_PATH.....	3959
SIM_COMPARE_SUMMARY	3959
SIM_COMPARE_X	3959
SIM_DELAY_SCAN_RELEASE	3960
SIM_DIAG_FILE	3961
SIM_DISTRIBUTED_PAT	3961
SIM_DUMPFILE_PATH	3962
SIM_EARLY_RELEASE.....	3962
SIM_EARLY_RELEASE_LAST.....	3963
SIM_EARLY_RELEASE_TIME.....	3963
SIM_FILL_BIDISCAN.....	3964
SIM_FORCE_Z_AT_CYCLE	3964
SIM_INCLUDE	3964
SIM_INSTANCE_NAME.....	3965
SIM_KEEP_PATH	3965
SIM_MASK_FILE.....	3966
SIM_MIN_NAME_FILE	3966
SIM_MISCOMPARE_LIMIT	3967
SIM_MODULE_INCLUDE	3967
SIM_MODULE_NAME	3967
SIM_NAME_FILE	3968
SIM_NO_CHAINNAME_FILE.....	3968
SIM_NO_INTERNAL_COMPARE	3969
SIM_NO_MEASURE_IN	3969
SIM_PARALLEL_DELAY_SHIFT_CLOCKS.....	3969
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE	3970
SIM_POST_SHIFT	3970
SIM_PRE_SHIFT	3971
SIM_PRECISION	3971
SIM_SERIAL_CHAIN	3971
SIM_SHIFT_DEBUG	3972
SIM_SIGNAL_SPY.....	3973
SIM_STATUS_MSG	3973
SIM_STRETCH_SCAN_RELEASE	3973

SIM_TIMEPLATE_COMM	3974
SIM_TMP_REG_LENGTH	3974
SIM_TOP_NAME	3974
SIM_VECTOR_COMM	3975
SIM_VECTYPE_SIGNAL	3976
STIL_2005_SCAN	3977
STIL_CHAIN_CELLS	3978
STIL_CHANNEL_CELLS	3978
STIL_CLOCK_WFC_PULSE	3979
STIL_DUAL_MODE	3979
STIL_FULL_CHAIN	3980
STIL_IDDQ_UPPERCASE	3980
STIL_MIN_QUOTE	3980
STIL_NESTED_MACRO	3981
STIL_NO_SCANSTRUCTURE	3981
STIL_NO_SCANX	3981
STIL_NOMEASURE_CLOCK	3982
STIL_PAT_ANN	3982
STIL_PAT_CMT	3983
STIL_PAT_LAB	3983
STIL_QUAD_MODE	3983
STIL_QUOTE_ALL	3984
STIL_SCAN_ANN1	3984
STIL_STRUCTURAL	3985
STIL_TI_TITLE	3987
STIL_TRIM_PULSE	3988
STIL_VECTOR_ANN	3988
STIL_VERG_ESC	3990
STIL_WRAP_ANNOTATION	3990
TITDL_CHAIN_SETTYPE	3990
TITDL_CUSTOMER	3991
TITDL_GROUP_TIMING	3991
TITDL_KEEP_SCANOUT	3991
TITDL_LIBRARY	3992
TITDL_PARTNUM	3992

TITDL_PSEUDO_PREFIX	3992
TITDL_REVISION	3993
TITDL_SET_DESCRIPTION	3994
TITDL_SETNAME	3994
TITDL_SETTYPE	3994
WGL_ADD_LASTX	3995
WGL_ALT_BIDI	3995
WGL_ALT_VECT_ANN	3996
WGL_EDGE_STROBE	3996
WGL_FULL_CHAIN	3997
WGL_FULL_SCANGROUP	3998
WGL_GROUP_PIN	3999
WGL_INV_SC	3999
WGL_NO_SCANX	4000
WGL_NOMEASURE_CLOCK	4000
WGL_ONE_ILLINOIS	4001
WGL_PATTERN_NAME	4001
WGL_TRIM.LEADING_P	4001
WGL_TRIM_PULSE	4002
WGL_VECTOR_ANN	4002
WGL_VERG_ESC	4003
WGL_VTRAN_PADSC	4003
WGL_WRAP_ANNOTATION	4003

A Note about the ALL_* Parameter File Keywords

All of the ALL_* parameter file keywords affect all pattern formats except for binary and ASCII.

ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS

This keyword applies the set_number_shifts specification to edt_internal patterns.

Usage

```
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS {0 | 1};
```

By default, when you use the `set_number_shifts` command with EDT on (compressed scan), the command specifies the number of external shifts used for loading the EDT channels (`write_patterns -edt_external`). However, when writing patterns using `-edt_internal`, the `set_number_shifts` specification does not apply. Setting `ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS` to 1 applies the `set_number_shifts` specification to any `edt_internal` patterns that are written. This applies to serial Verilog test benches and the other tester formats.

ALL_EXCLUDE_POWER_GROUND

This keyword excludes any port with a “function” value of “power” or “ground” from the written pattern.

Usage

```
ALL_EXCLUDE_POWER_GROUND {AUTO | 1 | 0};
```

By default, the keyword excludes such ports in tester formats, such as WGL and STIL, and does not exclude such ports for Verilog or VHDL. Setting this keyword to 1 excludes power and ground ports from all formats (except binary and ASCII); setting this keyword to 0 prevents exclusion of power and ground ports.

Note that binary and ASCII patterns are not affected by this keyword.

This parameter keyword is affected by the `include_all_power_pins` property in the Patterns/
[AdvancedOptions](#) wrapper of the [PatternsSpecification](#).

ALL_EXCLUDE_UNUSED

Removes certain pins from patterns.

Usage

```
ALL_EXCLUDE_UNUSED {0 | 1};
```

The default is 1 or true. The keyword removes the following pins from patterns:

- All output pins that are masked.
- All input pins that are constrained to CX and are not used in any test procedures.
- All bidi pins that are constrained to CX or CZ and are either not used in any test procedures or are only forced to Z and masked in the test procedures.

Note that this keyword does not remove pins from a multi-bit bus unless it can remove the entire bus.

Setting this keyword to 0 or false disables the pin removal.

This keyword is affected by the include_all_power_pins property in the Patterns/
AdvancedOptions wrapper of the [PatternsSpecification](#).

Keyword Usage with IJTAG Patterns

For IJTAG patterns, a primary port is not considered unused if one of the following applies:

- The port is an IJTAG test clock (TCKPort or CaptureEnPort with Attribute function_modifier = “CaptureShiftClock” or function_modifier = “CaptureShiftClockInv”).
- The port has been added as a clock by means of the [add_clocks](#) command.
- The port is relevant for the application of the retargeted PDL as follows:
 - ScanInterface ports
 - DataInPorts and DataOutPorts required for the retargeting of [iRead](#) and [iWrite](#) commands
 - All ports that have been subject to [iComparePort](#) or [iForcePort](#)
 - If [import_patterns_from_svf](#) has been used; all ports that are in the SVF PIOMAP
 - All BSDL compliance ports (ICL ports with attribute compliance_value)

ALL_FIXED_CYCLES

Issuing a 2 with this keyword causes the tool to compute the largest number of cycles between scan loads by looking at all patterns in the pattern list, not just those being saved.

Usage

```
ALL_FIXED_CYCLES {2 | 1 | 0};
```

Issuing a 2 with this keyword causes the tool to compute the largest number of cycles between scan loads by looking at all patterns in the pattern list, not just those being saved. Issuing a 1 with this keyword specifies that the patterns will contain an equal number of non-scan cycles between each scan chain loading. This non-scan cycle is applied before the first shift cycle. The default is 0.

This will add “Shift” cycles before the regular apply shift of the load_unload procedure. It adds these additional cycles before the scan chain load to make all patterns have the same total

number of cycles per pattern, taking into account the differences in the number of capture cycles. The number of capture cycles can still vary depending on what ATPG needed or was directed with the Named Capture Procedures. However, no additional “Capture” cycles are added. The new cycles will use the timeplate that the “Shift” procedure uses.

ALL_FLATTEN_TIMING

Issuing a 1 with this keyword causes all outputs to compute existing timing equations and use only the resulting numeric values in the output files.

Usage

```
ALL_FLATTEN_TIMING {1 | 0};
```

Issuing a 1 with this keyword causes all outputs to compute existing timing equations and use only the resulting numeric values in the output files. The default of 0 will result in any output format which can support equation-based timing using the equations as they are specified.

This keyword is intended for having pattern output files not contain equation-based timing that has been added to the procedure file and preserved in the tool. Test data languages such as Verilog, WGL, and STIL have the ability to express time values in the timing blocks, as numerical values or as equations based on variables. Using equation-based timing allows one value to be specified for a global attribute, such as the test cycle period, while other values are derived from this using equations.

ALL_FULL_TIMEPLATES

When this keyword is set to 1 or true, all of the timeplates that have been defined and stored in the tool will be written to the resulting pattern file or test bench.

Usage

```
ALL_FULL_TIMEPLATES {0 | 1};
```

Without this, only timeplates that are marked as used by the patterns in the pattern range being written are used. Similar to **SIM_NAME_FILE**, this should be used when writing out a Verilog test bench that can be used with one design but multiple pattern sets. Turning on **ALL_FULL_TIMEPLATES** will ensure that all timeplates that could be referenced in the *.vec* files in different pattern sets are enumerated in the test bench.

ALL_HEADER_COMMENT

Specifies comment lines to add to the header of the pattern file.

Usage

```
ALL_HEADER_COMMENT string;
```

The string specified is added as a comment line to the header of the pattern file. Use an embedded newline ‘\n’ character for the comment string to span multiple lines.

Suppose you want to add comments to the header of a verilog pattern file. Do this with the following command:

```
write_patterns results/pat.v -replace -profile -ext -verilog -parameter_list \
{ALL_HEADER_COMMENT "Header Line 1\nHeader Line 2"}
```

These lines are added to the pattern file header:

```
// Header Line 1
// Header Line 2
```

ALL_IDDQ_TESTER_CYCLE

Issuing a 1 with this keyword forces the IDDQ file to contain tester cycle counts and serial simulation times even if the pattern file being saved is a parallel load pattern file.

Usage

```
ALL_IDDQ_TESTER_CYCLE {1 | 0};
```

Issuing a 1 with this keyword forces the IDDQ file to contain tester cycle counts and serial simulation times even if the pattern file being saved is a parallel load pattern file. Not using the keyword, or using it with a 0 argument invokes the default behavior where the IDDQ file uses tester cycle counts for serial pattern formats, and vector counts and parallel load times for parallel formats.

In the example, saving a parallel load pattern file for a particular design and using the -iddq_file, the IDDQ file has the following content:

```
/* Pattern # Cycle # Time */
0 71 7100
1 76 7600
...
```

Saving the same parallel load pattern file and using -iddq_file, but also using ALL_IDDQ_TESTER_CYCLE 1; produces an IDDQ file with the following content:

```
/* Pattern # Cycle # Time */
0 195 19500
1 324 32400
...
...
```

The cycle counts and times given now correspond to serially loaded tester cycle counts and simulation time stamps

ALL_MAX_LOOP

Sets a maximum loop count to use for a single loop. Any run loops that are larger than this value are split into multiple smaller loops, with most being the size specified by ALL_MAX_LOOP and the last loop giving the remainder.

Note

 This keyword only affects loops that are run loops generated by the IJTAG solver either in an [iCall](#) within test_setup or test_end, or for the context pattern -ijtag for retargeted patterns. See also the [iRunLoop](#) command.

Usage

```
ALL_MAX_LOOP N
```

N is a long integer value specifying the maximum loop count. The minimum value is 4, and the maximum is 2^{40} . The default for most pattern formats is off, but for the Verilog testbench the default is 2^{30} .

If you specify an incorrect value, the tool issues a warning stating that the value is illegal, and that the keyword is being ignored.

```
// Warning: Parameter Keyword ALL_MAX_LOOP has an illegal value, being
// ignored.
```

This keyword does not affect repeats or loops that come from repeated sub_procedures or loop statements within the test procedure file. As with all other parameter keywords, this only affects the STI pattern formats, such as the Verilog testbench, STIL, or WGL.

Related Topics

[iRunLoop](#)

ALL_MIN_SCAN_LOAD

Setting this keyword to 1 allows a scan load to specify only scan-in or scan out when appropriate.

Usage

```
ALL_MIN_SCAN_LOAD {1 | 0};
```

By default, all pattern outputs contain scan-in and scan out data for each scan load. Setting this keyword to 1 allows a scan load to specify only scan-in or scan out when appropriate (first scan in). The default is 0.

ALL_NO_CYCLE_OPT

This keyword specifies for all outputs to turn off cycle optimizations in non-scan areas of patterns and leave all cycles in the patterns.

Usage

```
ALL_NO_CYCLE_OPT {1 | 0};
```

When set to 1 (or omitting this keyword as it is the default), this keyword specifies that cycle optimization is not applied to remove duplicate cycles from clock_sequential and capture procedures. When set to 0, this keyword allows the following optimization in cycles that are part of the Capture or Clock procedures: If any two identical cycles are generated and do not measure any outputs, the duplicate cycles will be removed.

ALL_NO_LOOP

Disables the use of “loops” or “repeat” statements to represent sub_procedures.

Usage

```
ALL_NO_LOOP {1 | 0};
```

Disables the use of “loops” or “repeat” statements to represent sub_procedures. The default is off (0). For more information, refer to “[Disabling Sub_procedure Looping](#)” in the *Tessent Shell User’s Manual*.

ALL_NO_PATTERN_TYPE

If set to 1, this keyword eliminates the pattern type annotation.

Usage

```
ALL_NO_PATTERN_TYPE {1 | 0};
```

If set to 1, this keyword eliminates the pattern type annotation. This keyword applies to all Vector Interface formats. The default is 0.

Caution

 This keyword should only be used if the pattern type annotations are causing problems. If you set this keyword to 1, you will not be able to load the generated patterns back into the Tesson tools.

The keyword should not be used with any STIL or WGL pattern file to be read back in as it removes the pattern type annotation at the start of each pattern that identifies a pattern as something other than a basic scan pattern. With the clock_po pattern type annotation removed, the code to extract the ASCII pattern data from the STIL pattern file does not identify this as a clock_po pattern and incorrectly puts the pulse clock statement in and the unload scan chain statement. All STI patterns always have overlapped scan load and unload, so the unload is always present in the pattern data.

ALL_NONSCAN_CONSTANT

Setting this keyword to 1 forces all non-scan pins, excluding control and clock pins, to the constant value of the pad state for all scan patterns.

Usage

```
ALL_NONSCAN_CONSTANT {1 | 0};
```

Setting this keyword to 1 forces all non-scan pins, excluding control and clock pins, to the constant value of the pad state for all scan patterns. This keyword may be useful, as an alternative to procedure force statements, for testers that require constant pin values. The default is 0.

ALL_ORIGINAL_INDEX

When set to 1, saved pattern files use the original pattern index (from before filtering) for the annotations and comments prior to each pattern.

Usage

```
ALL_ORIGINAL_INDEX {1 | 0};
```

When set to 1, saved pattern files use the original pattern index (from before filtering) for the annotations and comments prior to each pattern. When set to 0 (the default), the pattern numbering in the comments and annotations is numbered in order starting at 0 from the first pattern as if the filtered patterns are a new pattern set. Note that this affects all types of pattern outputs other than Verilog because the Verilog test bench uses a built-in pattern count mechanism.

ALL_PRESERVE_FORCE_X

This keyword retains or removes explicitly forced X values on ports in the pattern file.

Usage

```
ALL_PRESERVE_FORCE_X {1 | 0};
```

If X values are present on input pins in a vector, then previous force values for this vector are copied from previous vectors in place of this X value.

Setting this keyword to 0 or false (the default) means that the tool ignores the explicitly forced X values on ports in the pattern file.

Setting this keyword to 1 or true instructs the tool to retain explicitly forced X values on ports in the pattern file.

ALL_TIME_RESOLUTION

This keyword is used to specify the number of significant bits for floating point time numbers.

Usage

```
ALL_TIME_RESOLUTION integer;
```

This keyword is used to specify the number of significant bits for floating point time numbers. The maximum is 4, and the default is 3.

ALL_USE_CYCLE_LOOP

Uses updated terminology in the generated pattern files to prevent confusion with Tessent Diagnosis terms.

Usage

```
ALL_USE_CYCLE_LOOP {1 | 0};
```

Uses updated terminology in the generated pattern files to prevent confusion with Tessent Diagnosis terms. The previous “cycle” and “loop” terms in the pattern files are replaced with “simulation cycle” or “vector” and “tester cycle”. Set this keyword to 1 to revert back to the “cycle” and “loop” terminology.

CTL_ONE_PAT

This keyword, if set to 1, causes all pattern data to be grouped into a single STIL pattern block, instead of one pattern block per ATPG pattern. The default is 0.

Usage

```
CTL_ONE_PAT {1 | 0};
```

This keyword, if set to 1, causes all pattern data to be grouped into a single STIL pattern block, instead of one pattern block per ATPG pattern. The default is 0.

CTL_NO_PROTO

This keyword, if set to 1, disallows protocol-level macros.

Usage

```
CTL_NO_PROTO {1 | 0};
```

This keyword, if set to 1, disallows protocol-level macros. The default of 0 allows protocol-level macros.

Note

 In addition to the three CTL parameter file keywords listed above, all parameter file keywords for STIL output also work with CTL output. The tool-generated CTL output can be customized using the above three parameters to be a more structural STIL output.

CTL_STIL_0

This keyword, if set to 1, causes the output to be backward compatible with the IEEE 1450.6 standard.

Usage

```
CTL_STIL_0 {1 | 0};
```

This keyword, if set to 1, causes the output to be backward compatible with the IEEE 1450.6 standard. The Environment block will be dropped and the STIL header will not mention the CTL extension. The default is 0.

Note

 CTL (IEEE 1450.6) is an extension of STIL that creates a standard format to describe IP core and SOC test information.

CTL_STIL_1

This keyword, if set to 1, causes the output to eliminate all 1450.6 syntax, but leaves the IEEE 1450.1 syntax.

Usage

```
CTL_STIL_1 {1 | 0};
```

This keyword, if set to 1, causes the output to eliminate all 1450.6 syntax, but leaves the IEEE 1450.1 syntax. The default is 0, which produces the full 1450.6 and 1450.1 syntax used in the CTL output.

The CTL_STIL_0 parameter file keyword can be used in the CTL output to eliminate the 1450.1 features along with all of the 1450.6 syntax.

For more information, refer to the “[write_patterns -CTL](#)” command description in the *Tessent Shell Reference Manual*.

FTDL_DESIGNER

Specifies an informational string to be used as the designer name in the DESIGNER statement at the top of the FTDL file.

Usage

```
FTDL_DESIGNER string;
```

Specifies an informational string to be used as the designer name in the DESIGNER statement at the top of the FTDL file. If not specified, the default is “ATPG”.

FTDL_MAX_PAT_BLOCK

Specifies the maximum number of vector statements allowed in a single TEST block, including all shifted values and repeated vectors.

Usage

```
FTDL_MAX_PAT_BLOCK integer;
```

Specifies the maximum number of vector statements allowed in a single TEST block, including all shifted values and repeated vectors. If not specified, the default is 5 million.

FTDL_REVISION

Specifies an informational string to be used as the revision string in the REVISION statement at the top of the FTDL file.

Usage

```
FTDL_REVISION string;
```

Specifies an informational string to be used as the revision string in the REVISION statement at the top of the FTDL file. If not specified, the default is “0001”.

FTDL_TCOMMENT

Setting this keyword to a string allows the user to override the default TCOMMENT field in the FTDL pattern with a custom entry.

Usage

```
FTDL_TCOMMENT string;
```

FTDL_TEST_NAME

Specifies the root name to be used for the test block(s) within the FTDL file. If not specified, the default root name is “FN”.

Usage

```
FTDL_TEST_NAME string;
```

Specifies the root name to be used for the test block(s) within the FTDL file. If not specified, the default root name is “FN”. The root name is combined with a test number to create each test block name.

FTDL_TEST_NUM

Specifies the starting test block number, to be appended to the root test block name.

Usage

```
FTDL_TEST_NUM integer;
```

Specifies the starting test block number, to be appended to the root test block name. For each additional test block needed in the FTDL file, the test number is incremented. If not specified, the starting test block number is 1.

FTDL_TEST_SUFFIX

Specifies a string to add as a suffix to the test block name.

Usage

```
FTDL_TEST_SUFFIX string;
```

Specifies a string to add as a suffix to the test block name. The total test block name is constructed from FTDL_TEST_NAME, FTDL_TEST_NUM, and optionally, FTDL_TEST_SUFFIX. The required argument *string* is the suffix that will be added after the test block number. The default is to have no suffix.

FTDL_ZMODE_MES

Controls whether the FTDL file allows for Z values to be measured on output pins and output sides of bidi pins.

Usage

```
FTDL_ZMODE_MES {1 | 0};
```

Controls whether the FTDL file allows for Z values to be measured on output pins and output sides of bidi pins. If set to 0, then the “ZMODE = NOMES” statement will be placed in all TEST blocks within the FTDL file, and no Z values will be used on output pins. If set to 1, then the “ZMODE = MES” statement are placed in all TEST blocks, and Z values are allowed on output pins. The default is 0, NOMES.

Keywords FTDL_TEST_NAME and FTDL_TEST_NUM are used together to create the name(s) of the TEST block(s) within the FTDL file.

Keyword FTDL_MAX_VEC_BLOCK is used to set the maximum number of vector statements allowed in a single TEST block within the FTLD file.

Keywords FTDL_REVISION and FTDL_DESIGNER simply supply informational strings to be used within the FTDL file.

Keyword FTDL_ZMODE_MES controls whether ZMODE = MES, or ZMODE = NOMES, and is used in the FTDL file.

The following is an example parameter file for FTDL.

```
FTDL_TEST_NAME "MyTest";
FTDL_REVISION "001.1";
FTDL_DESIGNER "Smith";
FTDL_ZMODE_MES 1;
```

SIM_ANNOTATE QUIET

When set to 1, this keyword suppresses the Verilog test bench from displaying the annotations during simulation.

Usage

```
SIM_ANNOTATE QUIET {1 | 0};
```

When set to 1, this keyword suppresses the Verilog test bench from displaying the annotations during simulation. The default behavior is to display the annotations.

SIM_CHAIN_ERROR

Setting this parameter to 1 adds a register/signal per scan chain to the Verilog test bench.

Usage

```
SIM_CHAIN_ERROR {1 | 0};
```

Setting this parameter to 1 adds a register/signal per scan chain to the Verilog test bench. This signal may be monitored during simulation and goes to 1 when a mismatch is detected on the scan chain. The name of the register/signal is mgcdft_chainname_error where *chainname* is the chain name specified on the add_scan_chains command line. The default is 0.

SIM_CHANGE_PATH

Setting this parameter to 1 inserts code into the test bench that allows you to change the path to the files during simulation.

Usage

```
SIM_CHANGE_PATH {1 | 0};
```

Setting this parameter to 1 inserts code into the test bench that allows you to change the path to the files during simulation. Use the VERILOG PlusArgs keyword “`NEWPATH=`” to pass a new file path to test bench. The tool adds a path separator after the string from the `+NEWPATH` plusarg before the filename. For example:

```
vsim top_CngP_v_ctl +NEWPATH=./results -c -voptargs=+acc=npr \
-do "run -all"
```

Note

 SIM_CHANGE_PATH and **SIM_KEEP_PATH** are mutually exclusive.
SIM_CHANGE_PATH is ignored if SIM_KEEP_PATH is present.

SIM_COMPARE_SUMMARY

Writes a summary of compares in the test bench at the end of simulation.

Usage

```
SIM_COMPARE_SUMMARY {1 | 0};
```

Using this keyword causes the following statements to be displayed by the test bench at the end of simulation:

- Simulation finished at time *sim_time*
- Number of miscomparisons = *num*
- Number of 0/1 compares = *num*
- Number of Z compares = *num*

SIM_COMPARE_X

When set to 1, this parameter adds code to the test bench to not mask X states, but to compare for differences.

Usage

```
SIM_COMPARE_X {1 | 0};
```

When set to 1, this parameter adds code to the test bench to not mask X states, but to compare for differences. This functionality is available in both Serial and Parallel test benches. Issuing a 0, the default behavior, causes the test bench to not include these capabilities.

Note

 While issuing a mismatch if we expect X is not required for ATPG, it is necessary for LibComp verification to identify cases where the translation is pessimistic; otherwise, modeling a state such as a Tie-X will not look like a problem since you won't get mismatches.

SIM_DELAY_SCAN_RELEASE

This keyword and positive integer pair delays the scan release in the parallel test bench by the specified number.

Usage

```
SIM_DELAY_SCAN_RELEASE integer;
```

This keyword and positive integer pair delays the scan release in the parallel test bench by the specified number. The unit of the specified *integer* is the same as the timescale units. If there are multiple chains, each release is delayed by the specified *integer*.

By default, the parallel test bench uses the timing from the shift procedure to produce the force, release, and shift clock events. The release occurs at the end of the shift procedure. Since the parallel test bench forces the clock events at the device boundary, but the force and release events happen at the cell boundaries, clock propagation delay can cause the clock edges to arrive after the release. If the period of the shift procedure is not large enough to prevent the late arrival of the clock edges, you can specify an additional delay to the release using the **SIM_DELAY_SCAN_RELEASE** keyword, which adds to the period of the shift procedure for the parallel test bench only.

The **SIM_DELAY_SCAN_RELEASE** keyword cannot be specified with the **SIM_STRETCH_SCAN_RELEASE** keyword as the two keywords will cause two different mechanisms for delaying the scan release event to be used together. In the event both keywords are specified, the tool issues a warning that the keywords are mutually exclusive. In this case, the **SIM_STRETCH_SCAN_RELEASE** keyword will override the **SIM_DELAY_SCAN_RELEASE** keyword.

SIM_DIAG_FILE

Controls the generation of the failure file in the test bench.

Usage

```
SIM_DIAG_FILE {2 | 1 | 0};
```

Controls the generation of the failure file in the test bench. The failure file name is generated by appending the test bench file name with “.fail”. For example, a test bench file named *testbench.v* would generate a failure file named *testbench.v.fail*.

For uncompressed ATPG, this keyword works for both serial and parallel Verilog test benches. For compressed ATPG, the keyword only works for the serial Verilog test benches and has no impact on the parallel test bench.

For “patterns -scan” context, the default setting is 1. For all other contexts, the default is 0. The options for this parameter are as follows:

- **0** — No Verilog code for generating the failure file is included in the test bench. This setting maintains backward compatibility with previous test bench versions.
- **1** — Verilog code for generating the failure file is included in the test bench, but the file is not automatically generated. To activate generation of the failure file, you must edit the test bench, setting the “_write_DIAG_file” parameter to 1.
- **2** — Verilog code for generating the failure file is included and activated in the test bench. You can explicitly disable failure file generation by setting the “_write_DIAG_file” parameter to 0 in the test bench.

SIM_DISTRIBUTED_PAT

Turns off the enhanced Verilog test bench that allows you to choose which patterns to simulate.

Usage

```
SIM_DISTRIBUTED_PAT {1 | 0};
```

Turns off the enhanced Verilog test bench that allows you to choose which patterns to simulate. When set to 0, all saved patterns are simulated. When set to 1 (the default), this statement allows you to choose which patterns to simulate.

SIM_DUMPFILE_PATH

You can then use the STARTPAT and ENDPAT PlusArgs keywords to specify the starting and ending patterns. For example, the following runs the simulation using pattern files 2 through 10:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=2 +ENDPAT=10
```

If you specify only the ENDPAT keyword, the simulation begins with the first pattern or file and end at the specified pattern or file. For example, the following runs the simulation for the first four patterns:

```
vsim simple_testbench_v_ctl -c -do vsim.do +ENDPAT=4
```

If you specify only the STARTPAT keyword, the simulation starts with the specified pattern or file and end at the last pattern or file. For example, the following runs the simulation for pattern files 3 through 20, where 20 is the last pattern file:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=3
```

To simulate a single pattern, specify the same value both the start and end keywords. For example, the following runs the simulation for only pattern 8:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=8 +ENDPAT=8
```

SIM_DUMPFILE_PATH

Specifies the path of all dump files during Verilog simulation.

Usage

```
SIM_DUMPFILE_PATH pathname;
```

Specifies the path of all dump files during Verilog simulation.

SIM_EARLY_RELEASE

Designates the Verilog test bench to put the “release” statement immediately after the trailing edge of the first clock pulse in the shift procedure.

Usage

```
SIM_EARLY_RELEASE {1 | 0};
```

Designates the Verilog test bench to put the “release” statement immediately after the trailing edge of the first clock pulse in the shift procedure. Issuing a 0, the default behavior, will place the “release” statement at the end of the cycle.

SIM_EARLY_RELEASE_LAST

Used in conjunction with the SIM_EARLY_RELEASE_TIME keyword.

Usage

```
SIM_EARLY_RELEASE_LAST {1 | 0};
```

When multi-cycle shift procedures are present, by default **SIM_EARLY_RELEASE** causes the scan release to happen in the cycle that has the force_sci statement at the time specified. When the SIM_EARLY_RELEASE_LAST keyword is specified, the time specified by **SIM_EARLY_RELEASE_TIME** is applied to the last cycle in the multi-cycle shift procedure.

SIM_EARLY_RELEASE_TIME

Specifies exactly what time you want the Verilog test bench “release” statement to happen using the time scale specified in the procedure file.

Usage

```
SIM_EARLY_RELEASE_TIME integer;
```

Specifies exactly what time you want the Verilog test bench “release” statement to happen using the time scale specified in the procedure file. The integer value is a time value that is required to be in the same time scale as that used in the timeplate for the shift procedure. The time specified must be later than the time of the force_sci event, but not greater than the time of the end of the cycle that the force_sci event occurs in. The time is relative to the start of the cycle.

This keyword does not replace the SIM_EARLY_RELEASE keyword, but can be used as an alternative. If both the SIM_EARLY_RELEASE and the SIM_EARLY_RELEASE_TIME keywords are specified in the parameter file, then the SIM_EARLY_RELEASE_TIME keyword takes precedence. If neither keyword is used, then the default behavior of putting the “release” statement at the end of the cycle is still used.

Note

 The SIM_EARLY_RELEASE and SIM_EARLY_RELEASE_TIME statements are helpful when complex scan cells that use multiple clocks need the release to happen at a certain time prior to the end of the shift procedure, in order for the parallel load test bench to load the correct values. That is, if the forced value is held too long, a second clock pulse will latch in the wrong value.

SIM_FILL_BIDISCAN

Setting this keyword to 1 sets the bidirectional scan_input pins to the scan-in value of the first cell.

Usage

```
SIM_FILL_BIDISCAN {1 | 0};
```

By default, the bidirectional scan_input pins are set to “X” during parallel scan loading in the Verilog test bench. Setting this keyword to 1 sets the bidirectional scan_input pins to the scan-in value of the first cell.

SIM_FORCE_Z_AT_CYCLE

Controls when a bidi pin switches to a Z state in the Verilog test bench.

Usage

```
SIM_FORCE_Z_AT_CYCLE {1 | 0};
```

Controls when a bidi pin switches to a Z state in the Verilog test bench. The default behavior is that when a bidi pin is going to change to a Z value during a given simulation vector, then the Z is forced at time 0 for that vector. However, when you set the keyword value to 0, the Z value applies to the bidi pin at the force time specified in the timeplate rather than at time 0. Note that if the force time happens after the internal driver for the bidi pin turns on, bus contention is possible.

SIM_INCLUDE

Adds an include statement into the Verilog test bench after the comments and before the first module.

Usage

```
SIM_INCLUDE filename;
```

Adds an include statement into the Verilog test bench after the comments and before the first module. The contents of the filename specified must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The Default behavior is to not add any include statements.

SIM_INSTANCE_NAME

Specifies the instance name for your design, to be used within the test bench.

Usage

```
SIM_INSTANCE_NAME string;
```

Specifies the instance name for your design, to be used within the test bench.

For example, if your current design is simple_chain2 and you want to specify the instance name presy, this is the entry in the parameter file:

```
SIM_INSTANCE_NAME presy ;
```

And this is the test bench:

```
// top_module_name = simple_chain2_pat_par_v_ctl
module simple_chain2_pat_par_v_ctl;

simple_chain2 presy (.reset(reset), .clk(clk), .in1(in1),
    .scan_in1(scan_in1), .scan_en(scan_en), .out0(out0),
    .out1(out1), .out2(out2), .out3(out3), .all0(all0),
    .all1(all1), .wlout(wlout), .scan_out1(scan_out1));
```

SIM_KEEP_PATH

Causes the Verilog test bench to use the complete pathname entered on the command line to specify the pattern data file.

Usage

```
SIM_KEEP_PATH {1 | 0};
```

Causes the Verilog test bench to use the complete pathname entered on the command line to specify the pattern data file. Issuing a 0, the default behavior, causes the Verilog test bench to specify the pattern data file without any path information.



Note

SIM_CHANGE_PATH and **SIM_KEEP_PATH** are mutually exclusive.
SIM_CHANGE_PATH is ignored when **SIM_KEEP_PATH** is present.

SIM_MASK_FILE

Controls the generation of the mask file, which captures simulation mismatches, in the test bench.

Usage

```
SIM_MASK_FILE {2 | 1 | 0};
```

Controls the generation of the mask file, which captures simulation mismatches, in the test bench. The mask file name is generated by appending the test bench file name with “.mask”. For example, a test bench file named *testbench.v* would generate a mask file named *testbench.v.mask*.

For “patterns -scan” context, the default setting is 1. For all other contexts, the default is 0.

- **0** — No Verilog code for generating the mask file is included in the test bench. This setting maintains backward compatibility with previous test bench versions.
- **1** — Verilog code for generating the mask file is included in the test bench, but the file is not automatically generated. To activate generation of the mask file, you must edit the test bench, setting the “_write_MASK_file” parameter to 1.
- **2** — Verilog code for generating the mask file is included and activated in the test bench. You can explicitly disable mask file generation by setting the “_write_MASK_file” parameter to 0 in the test bench.

Note

 The simulation mismatches for the ChainTest patterns are recorded in the SIM_MASK_FILE, but they are commented out.

SIM_MIN_NAME_FILE

Controls the number of *.name* output files generated when saving patterns as a Verilog test bench.

Usage

```
SIM_MIN_NAME_FILE {1 | 0};
```

Controls the number of *.name* output files generated when saving patterns as a Verilog test bench. Setting this keyword to 0 causes the tool to generate a separate set of *.name* files for each test bench file. When using the -maxloads switch for large designs with many scan chains and patterns, this generates a very large number of files. Setting this keyword to 1 reduces the

number of output files by generating only one set of *.name* files for a design. The same *.name* files can be used for all generated Verilog test benches for that design.

This parameter file keyword is on by default when you use the [write_patterns](#) command with the -maxloads switch. It is off by default otherwise.

SIM_MISCOMPARE_LIMIT

Sets a limit in the Verilog test bench for the number of miscompares to allow before the simulation terminates.

Usage

```
SIM_MISCOMPARE_LIMIT integer;
```

Sets a limit in the Verilog test bench for the number of miscompares to allow before the simulation terminates. By default, the test bench runs the simulation until all saved patterns are simulated, regardless of how many miscompares happen. This keyword causes the simulation to abort upon reaching the miscompare limit.

SIM_MODULE_INCLUDE

Similar to the SIM_INCLUDE parameter, except it adds the include statement within the module definition for the test bench instead of after the comments and before the first module.

Usage

```
SIM_MODULE_INCLUDE filename;
```

Similar to the SIM_INCLUDE parameter, except it adds the include statement within the module definition for the test bench instead of after the comments and before the first module. The *filename* can be any valid file path. The contents of the specified file must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The default behavior is to not add any include statements.

SIM_MODULE_NAME

Specifies a string that overrides the module name of the DUT in the Verilog simulation test bench.

Usage

```
SIM_MODULE_NAME string;
```

SIM_NAME_FILE

Specifies a string that overrides the module name of the DUT in the Verilog simulation test bench.

SIM_NAME_FILE

Allow multiple test benches to use the same name files regardless of the name of the test bench.

Usage

```
SIM_NAME_FILE {1 | 0};
```

When writing out multiple test benches where only one is meant to be compiled and used, and the other data sets will be used (.cfg and .vec files) to simulate patterns using the one common test bench, this keyword should be used so that only one set of name files is written (*.po.name* and *.chain.name*) to save on disk space.

SIM_NO_CHAINNAME_FILE

This keyword controls the creation of chain name files. Setting this keyword to 1 prevents the generation of chain name files.

Usage

```
SIM_NO_CHAINNAME_FILE {1 | 0};
```

When saving a Verilog test bench, the tool, by default, generates a single chain name file with all the scan cell names. The names are loaded into one register in the test bench so that when mismatches occur during simulation, the test bench reports the mismatches by cell name, rather than by scan chain name and cell number. To avoid exceeding the limit of the register size, if there are more than two million scan cells, the tool default changes to automatically create one chain name file per scan chain.

When the chain name files are created, there is an increase in the number of files written, the disk space used, and the memory footprint of the Verilog test bench. This impacts simulation time when the design is large and when, due to memory constraints, the Verilog simulator needs to swap memory while running. The performance impact of chain name file creation is due to increased memory use rather than extra processing within the test bench during simulation, and influenced by design size and amount of memory available.

Setting this keyword to 1 prevents the tool from generating the chain name files. It also reduces the Verilog test bench memory footprint, the number of files written, and amount of data stored on the disk, and potentially speeds up simulation.

SIM_NO_INTERNAL_COMPARE

Enables or disables internal compares for the internal expect statements in the Verilog testbench.

Usage

```
SIM_NO_INTERNAL_COMPARE {1 | 0};
```

If there are expect statement in the procedure file on internal registers, then these expect statements will generate internal compares in the generated Verilog test bench. When this parameter is set to 1 or true, the generated Verilog testbench will not have internal compares for the internal “expect” statements.

The default is 0 or false.

SIM_NO_MEASURE_IN

In the Verilog test bench, when a value is being forced on a bidirectional pin (as in input mode), the test bench also tries to measure the same value on the bidirectional pin when a measure happens.

Usage

```
SIM_NO_MEASURE_IN {1 | 0};
```

In the Verilog test bench, when a value is being forced on a bidirectional pin (as in input mode), the test bench also tries to measure the same value on the bidirectional pin when a measure happens. Setting this keyword to 1 causes the test bench to not measure input values on the output side of bidirectional pins. The default is 0.

When you issue a “[write_patterns](#) -Verilog -mode LSI” command, the tool automatically sets SIM_NO_MEASURE_IN to 1.

SIM_PARALLEL_DELAY_SHIFT_CLOCKS

Delays the leading edge of all shift clocks in all shift procedures by the specified number of timescale units.

Usage

```
SIM_PARALLEL_DELAY_SHIFT_CLOCKS {integer};
```

Delays the leading edge of all shift clocks in all shift procedures by the specified number of timescale units. Any events occurring after the leading edge of the first shift clock is also

delayed to maintain the event order. The specified number must be a positive integer. If the specified delay causes any event to occur beyond the period of the timeplate, the tool ignores this keyword.

This keyword is valid only for Verilog parallel test bench generation.

SIM_PARALLEL_EARLY_FORCE_AND_MEASURE

Setting this keyword to 1 or true moves the measure_sco event to time 0 and the force_sci event to time 1.

Usage

```
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE {1 | 0 | true | false};
```

Setting this keyword to 1 or true moves the measure_sco event to time 0 and the force_sci event to time 1. By default (0 or false), the measure_sco event occurs at the measure_po time specified in the timeplate for the shift procedure. The force_sci event then occurs one timescale unit later, regardless of the force_pi time specified in the timeplate.

This keyword is valid only for Verilog parallel test bench generation.

SIM_POST_SHIFT

Specifies the absolute number of independent post shifts to use during simulation.

Usage

```
SIM_POST_SHIFT integer;
```

Specifies the absolute number of independent post shifts to use during simulation. That is, if the load_unload procedure already has one post shift, setting this keyword to “2” causes two post shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of post shifts to the total number of shifts minus one.

You can use this keyword to generate a parallel simulation test bench that applies a parallel load followed by N number of serial shift cycles. When used in the “patterns -scan” context, SIM_POST_SHIFT serial shift cycles are applied by forcing the scan-in values onto the scan pin of the design. You can also use this keyword to generate an EDT parallel simulation test bench; in this case, the serial shifts are performed on the EDT internal scan chains and the test bench forces values onto the scan ports driven by the decompressor.

However, you cannot use this keyword to generate an EDT parallel simulation test bench in which the serial inputs are shifted in through the EDT channels because serial simulation through the decompressor and other EDT logic is not supported by parallel test bench.

SIM_PRE_SHIFT

Specifies the absolute number of independent pre shifts to the load_unload procedure (prior to main shift) to use during simulation.

Usage

```
SIM_PRE_SHIFT integer;
```

Specifies the absolute number of independent pre shifts to the load_unload procedure (prior to main shift) to use during simulation. That is, if the load_unload procedure already has one pre shift, setting this keyword to “2” causes two pre shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of pre shifts to the total number of shifts minus one.

SIM_PRECISION

Specifies the timescale precision.

Usage

```
SIM_PRECISION string;
```

Specifies the timescale precision. The string is a number and a unit, for example:

```
SIM_PRECISION 1ns;  
SIM_PRECISION 10ms;
```

The units supported are ms, us, ns, ps, and fs.

The tool checks this value to verify that it is less than or equal to the Timescale unit from the procedure file and outputs this value in the test benches.

SIM_SERIAL_CHAIN

Creates a test bench that applies chain test patterns in serial mode and all other patterns in parallel mode (when set to 1).

Usage

```
SIM_SERIAL_CHAIN {1 | 0};
```

Creates a test bench that applies chain test patterns in serial mode and all other patterns in parallel mode (when set to 1). The keyword is set to 0 by default.

SIM_SHIFT_DEBUG

Specifies for the serial Verilog test bench to use special test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the patterns.

Usage

```
SIM_SHIFT_DEBUG integer;
```

Specifies for the serial Verilog test bench to use special test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the patterns.

For debugging purposes only, this format makes it easier to pinpoint precisely which scan cell in a chain is causing a problem. Issuing a 0, which is the default behavior, causes the serial Verilog test bench to use test patterns that do not contain a parallel observe of the entire scan chain for every shift.

An integer value of 1 causes the test bench to shift serially and do a parallel compare for all shifts. An integer value of 2 or greater causes the test bench to shift serially and do parallel compares on the first 2 or greater shifts. This is done to reduce the amount of data in the pattern files.

The Verilog patterns you create using the keyword SIM_SHIFT_DEBUG typically take more time for the tool to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be huge). Also, because they simulate serially, the patterns take more time to simulate. Be sure these are acceptable trade-offs before trying to use these patterns.

The tool ignores the SIM_SHIFT_DEBUG keyword unless you include the “[write_patterns -Serial](#)” switch. Also, if you do not include the -End or “-pattern_sets chain” options, the tool defaults to “-End 0”, saving chain test patterns and the first scan pattern. To save chain test patterns and a different set of scan patterns, use the -Begin and/or -End switches to specify the range of scan test patterns to save. To save only chain test patterns, include the “-pattern_sets chain” option.

Note that this functionality is available only for patterns created in the “patterns -scan” context and EDT bypass patterns (uncompressed).

SIM_SIGNAL_SPY

Specifies to use the SignalSpy functions \$signal_force, \$signal_release, and \$init_signal_spy (for observing the scan outs) to do a parallel load of the scan chain (when set to 1).

Usage

```
SIM_SIGNAL_SPY {1 | 0};
```

Specifies to use the SignalSpy functions \$signal_force, \$signal_release, and \$init_signal_spy (for observing the scan outs) to do a parallel load of the scan chain (when set to 1). There are three integer variables in the test bench that you can change from “0” to “1” to get verbose messaging from the SignalSpy functions. The variables are _ssi_verbose for \$init_signal_spy, _ssf_verbose for \$signal_force, and _ssr_verbose for \$signal_release. The default is 0, which specifies to use the standard Verilog force and release to do a parallel load of the scan chain.

These functions work only with the ModelSim Verilog simulator. The SignalSpyPathSeparator in the modelsim.ini file must not be set or be set to the default of “/”.

SIM_STATUS_MSG

Specifies the frequency of status messages output from the Verilog simulation test benches.

Usage

```
SIM_STATUS_MSG integer;
```

Specifies the frequency of status messages output from the Verilog simulation test benches. This keyword accepts an integer that specifies the interval (number of patterns) to wait before outputting a message. The simulation message is “Simulated *n* patterns.”

SIM_STRETCH_SCAN_RELEASE

Delays the release scan event in the parallel Verilog testbench by stretching the end of the shift cycle.

Usage

```
SIM_STRETCH_SCAN_RELEASE integer;
```

This keyword and positive integer pair delays the release scan event in the parallel Verilog testbench by stretching the end of the shift cycle by the amount specified by the *integer* argument to the keyword. The unit of the specified *integer* is the same as the timescale units. If there are multiple chains, each release scan event is delayed by the specified *integer*.

The SIM_STRETCH_SCAN_RELEASE keyword cannot be used at the same time as SIM_DELAY_SCAN_RELEASE as the two keywords will cause two different mechanisms for delaying the scan release event. In the event both keywords are specified, the tool issues a warning that the keywords are mutually exclusive, and, in this case, the SIM_STRETCH_SCAN_RELEASE keyword will override the SIM_DELAY_SCAN_RELEASE keyword.

SIM_TIMEPLATE_COMM

Turns on timeplate comments in the vector file.

Usage

```
SIM_TIMEPLATE_COMM {1 | 0};
```

Turns on timeplate comments in the vector file. The default is off (0). When you set this parameter to 1, the tool adds comments for the timeplate to the vector file. The format of the timeplate comment is:

```
// Timeplate: <timeplate_name>
```

SIM_TMP_REG_LENGTH

Changes the default length of the NEWPATH variable used in the generated testbench.

Usage

```
SIM_TMP_REG_LENGTH integer;
```

Overrides the default (512) limit for the NEWPATH variable used for filenames in the testbench.

If you specify a value less than the path of the generated testbench, the tool ignores this value and issues a warning similar to the following:

```
// Warning: Ignored SIM_TMP_REG_LENGTH of 12, because its value cannot be
// less than 54, the length of the file name
// 'results/user_specified_path_length_that_is_too_short.v'.
```

SIM_TOP_NAME

Specifies a new top level module name for the Verilog test bench.

Usage

```
SIM_TOP_NAME string;
```

Specifies a new top level module name for the Verilog test bench. The default behavior is to generate a default top level module name based on the design name and the name of the pattern file.

SIM_VECTOR_COMM

Set this keyword to 1 to turn on vector type comments in the vector file, the default is off.

Usage

```
SIM_VECTOR_COMM {1 | 0};
```

Set this keyword to 1 to turn on vector type comments in the vector file, the default is off. The format of the vector type comment is:

```
// Vector type: vector_type_string
```

Where *vector_type_string* is one of the vector types listed in [Table A-1](#).

Table A-1. Vector Types

Vector Type	Definition
TEST_SETUP	Vector from test_setup procedure.
TEST_END	Vector from test_end procedure.
SEQ_TRANSPARENT	Vector from seq_transparent procedure.
CLOCK PROCEDURE	Vector from clock procedure.
CLOCK PROCEDURE RESTORE	Vector from clock procedure that uses restore_bidi event.
LOAD_UNLOAD	Vector from load_unload procedure.
SHIFT	Vector from shift procedure.
SINGLE_SHIFT	Single shift vector applied in load_unload procedure.
SHIFT_EXTRA	Vector from multicycle shift procedure that does not contain the force_sci and measure_sco events.
SHADOW_CONTROL	Vector from shadow_control procedure.
MASTER_OBSERVE	Vector from master_observe procedure.
SHADOW_OBSERVE	Vector from shadow_observe procedure.
SKEW_LOAD	Vector from skew_load procedure.

Table A-1. Vector Types (cont.)

Vector Type	Definition
PRIMARY ¹	Vector from a non-scan procedure such as capture, named capture, clock_sequential, or ram_sequential.
PRIMARY_MEASURE ¹	Vector from a non-scan procedure that measures primary outputs.
PRIMARY_CLOCKPO ¹	Vector from a clock_po procedure.

1. In addition to the above, the “PRIMARY” vector types can have “_IDQ” appended to the end to indicate an idq measure point.

SIM_VECTYPE_SIGNAL

When set to 0, specifies that all vector types in the currently active procedure be represented in the pattern file as a signal, and that these signals be “1” when that vector type is active.

Usage

```
SIM_VECTYPE_SIGNAL {1 | 0 | 2};
```

When set to 0, specifies that all vector types in the currently active procedure be represented in the pattern file as a signal, and that these signals be “1” when that vector type is active. This allows you to specify the patterns for tracing and viewing so you can see what type of vector is currently active at any specified moment within a simulation. For an example of using this keyword, refer to “[Simulation Data Analysis](#)” in the *Tessent Scan and ATPG User's Manual*.

When set to 1, this keyword adds an encoded vector type to the Verilog or VHDL test bench in the bus MGCDFT_VECTYP[0-3]. [Table A-2](#), lists the signal or variable names added to the test bench that map to the bus encoding. Setting the keyword to 0 causes the test bench to not include these capabilities.

Table A-2. Translation Table

MGCDFT_VECTYP[0-3] Encoding	Signal/Variable
0001	mgcdft_test_setup
0010	mgcdft_load_unload
0011	mgcdft_shift
0100	mgcdft_single_shift
0101	mgcdft_shift_extra
0110	mgcdft_shadow_control
0111	mgcdft_master_observe
1000	mgcdft_shadow_observe

Table A-2. Translation Table (cont.)

MGCDFT_VECTYP[0-3] Encoding	Signal/Variable
1001	mgcdft_skew_load
1010	mgcdft_seq_transparent
1011	mgcdft_launch_capture
1100	mgcdft_shift
1101	mgcdft_clock_proc
1110	mgcdft_clock_proc_restore
1111	mgcdft_test_end
0000	mgcdft_unknown

Note

 Adding signals indicating vector type to the Verilog test bench enables you to see what type of vector is currently active at any specified moment within a simulation and can simplify simulation mismatch debugging.

Setting this keyword to 2 (the default), which is available only with a Verilog test bench, also adds the encoded vector types (default behavior). Additionally, the test bench will have the following _procedure_strings, which can be added to your WAVE display to show the currently active Procedure Type.

CLOCK_PROC	SEQ_TRANSPARENT	SINGLE_SHIFT
CLOCK_PROC_RES	SHADOW_CONTROL	SKEW_LOAD
LAUNCH_CAPTURE	SHADOW_OBSERVE	TEST_SETUP
LOAD	SHIFT	TEST_END
MASTER_OBSERVE	SHIFT_EXTRA	UNKNOWN

STIL_2005_SCAN

Issuing a 1 with this keyword is similar to the STIL_STRUCTURAL keyword except it also uses the IEEE 1450.1 ScanStructures block.

Usage

```
STIL_2005_SCAN {1 | 0};
```

Issuing a 1 with this keyword is similar to the STIL_STRUCTURAL keyword except it also uses the IEEE 1450.1 ScanStructures block. The default is 0, which produces the existing IEEE 1450.0 ScanStructures block.

With the STIL_2005_SCAN keyword, the CTL output uses the 1450.1 ScanStructures block by default. You can use the CTL_STIL_0 parameter file keyword in the CTL output to eliminate the new 1450.1 features along with all of the 1450.6 syntax. The parameter file keyword CTL_STIL_1 can be used to eliminate all 1450.6 syntax but leave the 1450.1 syntax.

Note that the ScanStructures in your STIL structure must match the compressed structure.

In order to use this output, you must use the “[write_patterns -STIL1999](#)” command.

STIL_CHAIN_CELLS

Setting this keyword to 0 removes the ScanCells list from the ScanChain definition in the ScanStructures block when saving non-compressed scan patterns that use scan chains.

Usage

```
STIL_CHAIN_CELLS {1 | 0};
```

Setting this keyword to 0 removes the ScanCells list from the ScanChain definition in the ScanStructures block when saving non-compressed scan patterns that use scan chains. The following example shows a ScanStructures block without the ScanCells list:

```
ScanStructures {
    ScanChain chain1 {
        ScanLength 100;
        ScanInversion 0;
        ScanIn "sci";
        ScanOut "sco";
    }
}
```

By default (1), the scan cell names are listed following the ScanLength statement.

For compressed patterns, see the STIL_CHANNEL_CELLS keyword.

STIL_CHANNEL_CELLS

Enables the listing of dummy scan cell names in the ScanStructures block when saving compressed patterns that use EDT channels.

Usage

```
STIL_CHANNEL_CELLS {1 | 0};
```

Enables the listing of dummy scan cell names in the ScanStructures block when saving compressed patterns that use EDT channels. By default (0), the dummy scan cell names are not listed.

For non-compressed patterns, see the STIL_CHAIN_CELLS keyword.

STIL_CLOCK_WFC_PULSE

This parameter, used with the write_patterns command, controls the waveform characters of a clock signal in the generated STIL pattern file.

Usage

```
STIL_CLOCK_WFC_PULSE {TRUE | FALSE};
```

When STIL_CLOCK_WFC_PULSE is FALSE or not set at all, the waveform character set for clock signals is “01”. For a positive pulse, “1” indicates the clock signal is ON for the cycle and “0” is OFF for the cycle.

```
Timing STUCK_timing {
    WaveformTable tset_tp_shift_capture {
        Period '32ns' ;
        Waveforms {
            PI_grp_1 { 01 { '0ns' D; '10ns' D/U; '22ns' D; }}// positive pulse
            PI_grp_2 { 01 { '0ns' U; '10ns' D/U; '22ns' U; }}// negative pulse
        }
    }
}
```

When STIL_CLOCK_WFC_PULSE is TRUE, the waveform character set for clock signals is “0P” or “N1”. The waveform character set for a positive pulse is “0P”. The “P” indicates the clock signal is ON and “0” is OFF for the cycle.

The waveform character set for a negative pulse is “N1”. The “N” indicates the clock signal is ON and “1” is OFF for the cycle.

```
Timing STUCK_timing {
    WaveformTable tset_tp_shift_capture {
        Period '32ns' ;
        Waveforms {
            PI_grp_1 { 0P { '0ns' D; '10ns' D/U; '22ns' D; }}// positive pulse
            PI_grp_2 { N1 { '0ns' U; '10ns' D/U; '22ns' U; }}// negative pulse
        }
    }
}
```

STIL_DUAL_MODE

Enables dual mode support for IJTAG patterns that are in the STIL format.

Usage

```
STIL_DUAL_MODE {1 | 0};
```

Turns on (1) dual mode support for IJTAG patterns that are in the STIL format.

You use this parameter to ensure the IJTAG STIL patterns support dual-mode, where two vectors can be executed by one ATE cycle. To support dual mode, the number of vectors before the loop, inside the loop, between loops, and after the loop must be a multiple of two.

To support this requirement, the tool may partially or fully unroll loops and add padding to the end of the test. Added padded vectors have all outputs masked to X and all clocks turned off to ensure that they do not affect the timing or functionality of the test.

To set to a multiple of 4, see the [STIL_QUAD_MODE](#) parameter keyword.

STIL_FULL_CHAIN

Causes the tool to shorten the list of scan cells by the number of pre shifts and post shifts.

Usage

```
STIL_FULL_CHAIN {1 | 0};
```

Causes the tool to shorten the list of scan cells by the number of pre shifts and post shifts. By default (1), all cells in the scan chain are listed in the STIL file regardless the number of pre shifts and post shifts.

STIL_IDDQ_UPPERCASE

If this is set to 1 or true, IDDQ test points in the STIL file will use the term IDDQTestPoint, if 0 or false, which is the default, the STIL file will use IddqTestPoint.

Usage

```
STIL_IDDQ_UPPERCASE {1 | 0};
```

Some STIL parsers will only accept one or the other.

STIL_MIN_QUOTE

Issuing a 1 with this keyword is for STIL2005 and CTL outputs, and causes the tool to write the STIL file using minimum quoting of signal names that use reserved characters.

Usage

```
STIL_MIN_QUOTE {1 | 0};
```

Issuing a 1 with this keyword is for STIL2005 and CTL outputs, and causes the tool to write the STIL file using minimum quoting of signal names that use reserved characters. Further, the existing STIL1999 output can be changed to use the same minimum quoting rules. The default is 0 for the STIL1999 output.

STIL_NESTED_MACRO

Setting this keyword to 1 when saving STRUCTURAL_STIL, STIL2005, or CTL patterns causes sub procedures to be interpreted as Macros. As Macros, they are nested inside the calling Macros.

Usage

```
STIL_NESTED_MACRO {1 | 0};
```

Setting this keyword to 1 when saving STRUCTURAL_STIL, STIL2005, or CTL patterns causes sub procedures to be interpreted as Macros. As Macros, they are nested inside the calling Macros. The tool uses loop statements, if necessary, to repeat the Macro. This keyword is set to 0 by default.

STIL_NO_SCANSTRUCTURE

Setting this keyword to 1 removes the entire ScanStructures block from the STIL output.

Usage

```
STIL_NO_SCANSTRUCTURE {1 | 0};
```

Setting this keyword to 1 removes the entire ScanStructures block from the STIL output. For design flows where the ScanStructures block is not required, removing the block significantly reduces the size of the STIL file. The ScanStructures block is included by default (0).

STIL_NO_SCANX

If set to 1, this keyword replaces all X scan load values in the STIL pattern output with 0 scan load values. The default is 0.

Usage

```
STIL_NO_SCANX {1 | 0};
```

If set to 1, this keyword replaces all X scan load values in the STIL pattern output with 0 scan load values. The default is 0.

STIL_NOMEASURE_CLOCK

Issuing a 1 with this keyword drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured.

Usage

```
STIL_NOMEASURE_CLOCK {1 | 0};
```

Issuing a 1 with this keyword drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. To deactivate, issue a 0 (the default) with this keyword.

Examples follow:

```
Parameter file : default
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
          01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
                  STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '200ns' U/D; '300ns' D } }
```

STIL_PAT_ANN

Specifies the annotation string that the tool outputs to the STIL pattern file just prior to each scan pattern.

Usage

```
STIL_PAT_ANN string %special_keyword string2;
```

Specifies the annotation string that the tool outputs to the STIL pattern file just prior to each scan pattern.

```
annotation "string %keywords string"
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Parameter file entry:

```
STIL_PAT_ANN  "Pattern Number:%pat_num  
(%pat_num_plus1), Cycle Number:%cycle_num"
```

Resultant pattern file statement:

```
Ann { * "Pattern Number:0 (1), Cycle Number:0" * }
```

STIL_PAT_CMT

Turns off (0) the pattern comment that precedes each ATPG pattern. The default is 1.

Usage

```
STIL_PAT_CMT {1 | 0};
```

Turns off (0) the pattern comment that precedes each ATPG pattern. The default is 1.

STIL_PAT_LAB

Turns off (0) the pattern label that precedes each ATPG pattern.

Usage

```
STIL_PAT_LAB {1 | 0};
```

Turns off (0) the pattern label that precedes each ATPG pattern. The default is 1.

STIL_QUAD_MODE

Enables quad mode support for IJTAG patterns that are in the STIL format.

Usage

```
STIL_QUAD_MODE { 1 | 0};
```

Turns on (1) quad mode support for IJAG patterns that are in the STIL format.

STIL_QUOTE_ALL

You use this parameter to ensure the IJTAG STIL patterns support quad-mode, where four vectors can be executed by one ATE cycle. To support quad mode, the number of vectors before the loop, inside the loop, between loops, and after the loop must be a multiple of four.

To support this requirement, the tool may partially or fully unroll loops and add padding to the end of the test. Padded vectors have all output masked to X and all clocks turned off to ensure that they do not affect the timing or functionality of the test.

To set to a multiple of 2, see the [STIL_DUAL_MODE](#) parameter keyword.

STIL_QUOTE_ALL

Causes all user identifiers to be enclosed in double quotes in the STIL output file.

Usage

```
STIL_QUOTE_ALL {1 | 0};
```

Causes all user identifiers to be enclosed in double quotes in the STIL output file. By default (0), user identifiers are not quoted.

If the STIL_QUOTE_ALL and STIL_MIN_QUOTE keywords are both active (1), the tool ignores the STIL_MIN_QUOTE keyword.

STIL_SCAN_ANN1

Specifies the string the tool outputs to the STIL pattern just prior to the first scan pattern.

Usage

```
STIL_SCAN_ANN1 string %special_keyword string2;
```

Specifies the string the tool outputs to the STIL pattern just prior to the first scan pattern. There is only one STIL_SCAN_ANN1 statement written for each pattern file generated.

```
annotation "string %keywords string"
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Parameter file entry:

```
STIL_SCAN_ANN1 "Scan Chain:%chain, length:%length,  
SI:%scan_in, SO:%scan_out"
```

Resultant pattern file statement:

```
Ann {*  
"Scan Chain:chain1, length:2, SI:scan_in1, SO:Q" *}
```

STIL_STRUCTURAL

Causes the STIL file to be written using a more structural form.

Usage

```
STIL_STRUCTURAL {1 | 0};
```

Causes the STIL file to be written using a more structural form. This keyword set to 1 results in a STIL output form that uses some of the basic hierarchical approaches of CTL output, but not any of the IEEE 1450.6 CTL syntax extensions. As with CTL output, the structural STIL output creates STIL Macro definitions for all used procedures, and a STIL Procedure definition for the test_setup procedure, if present. The start of each pattern still consists of an annotation statement and a pattern label. Each pattern consists of the calls to each specific Macro, passing all needed data. Unlike the CTL output, a new “pattern type” macro will not be created for each pattern, and the pattern data will not be placed in a separate file and included with the “Include” statement.

To produce a flat or non-hierarchical set of vector data, set this keyword to 0 (default).

The following is an example of the structural STIL output:

```
STIL 1.0 ;

Header { ... }

// Parameter File Keyword Settings
// STIL_STRUCTURAL true ;

Signals {
    "IN"[3..0] In; "CLK" In; "SDI" In; "SE" In;
    "OUT"[3..0] Out; "SDO" Out;
}

SignalGroups {
    _pi_[6..0] = '"IN"[3..0] + "CLK" + "SDI" + "SE"';
    _po_[4..0] = '"OUT"[3..0] + "SDO"';
    PI_grp_0[5..0] = '"IN"[3..0] + "SDI" + "SE"';
    PI_grp_1 = '"CLK"';
    "_chain_SDI_" = '"SDI"' {ScanIn 10;};
    "_chain_SDO_" = '"SDO"' {ScanOut 10;};
}

Timing TRANSITION_timing {
    WaveformTable tset_tp_shift {
        Period '400ns';
        Waveforms {
            PI_grp_0 { 01X { '0ns' D/U/N; } }
            PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D; } }
            _po_ { LHXZ { '0ns' X; '50ns' 1/h/X/t; '52ns' X; } }
        }
    }
    WaveformTable tset_tp_capture {
        Period '400ns';
        Waveforms {
            PI_grp_0 { 01X { '0ns' D/U/N; } }
            PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D; } }
            _po_ { LHXZ { '0ns' X; '50ns' 1/h/X/t; '52ns' X; } }
        }
    }
}

ScanStructures { ScanChain chain { ... } }

Procedures {
    test_setup {
        W tset_tp_shift;
        V { "SE" = 0; }
    }
}

MacroDefs {
    load_unload_grp1 {
        W tset_tp_shift;
        V { "CLK" = 0; }
        Shift { V { "SE" = 1; "_chain_SDI_" = #;
                    "_chain_SDO_" = #; "CLK" = 1; } }
    }
}
```

```
        }

    capture {
        W tset_tp_shift;
        V { _pi_ = %; _po_ = %; PI_grp_1 = %; }
    }

PatternBurst scanpats { PatList { scan_test } }

PatternExec scanexec {
    Timing TRANSITION_timing;
    PatternBurst scanpats;
}

Pattern "scan_test" {
    Call test_setup ;
    Ann {* Begin chain test *}
    Ann {* Chain Pattern:0 Cycle:1 Loop:1 *}
    ...
    Ann {* End chain test *}
    Ann {* Pattern:0 Cycle:7 Loop:10 *}
    "pattern 0": Macro "load_unload_grp1" {
        "_chain_SDI_" = 1010001110;
        "_chain_SDO_" = XXXXXXXXXXXX;
    }
    Macro "capture" {
        _pi_ = X101011;
        _po_ = LLHHH;
        PI_grp_1 = 1;
    }

    ...
    Ann {* Pattern:7 Cycle:24 Loop:52 *}
    "pattern 0": Macro "load_unload_grp1" {
        "_chain_SDI_" = 1010010000;
        "_chain_SDO_" = HHLHHLLLLL;
    }

    Macro "capture" {
        _pi_ = X000011;
        _po_ = LHHLL;
        PI_grp_1 = 1;
    }

    last_unload: Macro "load_unload_grp1" {
        "_chain_SDI_" = 1010010000;
        "_chain_SDO_" = LHLLLHLLLH;
    }
}
```

STIL_TI_TITLE

Turns on a TI-specific title block in the STIL Header block.

Usage

```
STIL_TI_TITLE {1 | 0};
```

Turns on a TI-specific title block in the STIL Header block. This title block contains all of the specific statements from the TITDL output file, and you can use the TITDL-specific keywords to change the fields in the title block. The default is to turn off the title block.

The TITDL-specific keywords include: TITDL_LIBRARY, TITDL_CUSTOMER, TITDL_PARTNUM, TITDL_SETNAME, TITDL_SETTYPE, TITDL_CHAIN_SETTYPE, and TITDL_REVISION.

STIL_TRIM_PULSE

Causes the input timing to drop the leading 0ns force to the off state of the clock.

Usage

```
STIL_TRIM_PULSE {1 | 0};
```

Causes the input timing to drop the leading 0ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. To deactivate, set this keyword to 0 (default).

Examples follow:

```
Parameter file : default
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
          01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
          01 { '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
                  STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '200ns' U/D; '300ns' D } }
```

STIL_VECTOR_ANN

Specifies the vector types to be added to the output pattern file.

Usage

```
STIL_VECTOR_ANN string %special_keyword string2;
```

Specifies the vector types to be added to the output pattern file.

```
annotation "string %keywords string"
```

Note

 You must not use the “%” symbol as the leading character to any string fragment in conjunction with the %special_keyword.

The STIL_SCAN_ANN1, STIL_PAT_ANN, and STIL_VECTOR_ANN parameter file keywords use a (%special_keyword) that is translated at the specific time and place the annotation statement is written. [Table A-3](#) lists the available special keywords. [Table A-4](#) lists the pattern types. [Table A-5](#) lists the vector types.

Table A-3. STIL Special Keywords

Keyword	Description
%chain	The chain name of the current chain.
%cycle_num	The current ATPG cycle number.
%length	The length of the current chain.
%pat_num	The current ATPG pattern number.
%pat_num_plus1	The current ATPG pattern number plus 1.
%pattern_type	The pattern type, for outputting annotations in the pattern file. It causes one of the pattern types in Table A-4 to be written into the pattern file.
%scan_in	The name of the Primary Input pin for the current chain.
%scan_out	The name of the Primary Output pin for the current chain.
%vector_type	This keyword causes one of the vector types in Table A-5 to be written into the pattern file.

Table A-4. STIL Pattern Types

basic	macrotest	ram_passthru
clock_po	multi_load	ram_sequential
clock_sequential		

Table A-5. STIL Vector Types

CLOCK_PROCEDURE	PRIMARY_CLOCKPO	SHADOW_CONTROL
CLOCK_PROCEDURE_RESTORE	PRIMARY_CLOCKPO_IDDQ	SHADOW_OBSERVE
CORE_ACCESS	PRIMARY_IDDQ	SHIFT

Table A-5. STIL Vector Types (cont.)

CORE_INST_ISOLATE	PRIMARY_MEASURE	SINGLE_SHIFT
CORE_INST_TEST	PRIMARY_MEASURE_IDDQ	SKEW_LOAD
LOAD_UNLOAD	SCAN_IN	SEQ_TRANSPARENT
MASTER_OBSERVE	SCAN_IO	TEST_SETUP
PRIMARY	SCAN_OUT	

STIL_VERG_ESC

Setting this keyword to 1 changes all vectored signal names to scalars.

Usage

```
STIL_VERG_ESC {1 | 0};
```

Setting this keyword to 1 changes all vectored signal names to scalars. For example, the pin name “bus”[0] would change to “bus[0]”.

STIL_WRAP_ANNOTATION

Breaks multi-line annotations into multiple annotation statements.

Usage

```
STIL_WRAP_ANNOTATION {1 | 0};
```

When this keyword is set to 1 or true in the parameter file, then any multi-line annotations are broken into multiple annotation statements, such as:

```
Ann /* this is a long */  
Ann /* multi-line annotation */
```

If not present in the parameter file, the default for this keyword is 0 or off.

TITDL_CHAIN_SETTYPE

Specifies the string used to set the chain type.

Usage

```
TITDL_CHAIN_SETTYPE string;
```

Specifies the string used to set the chain type. If specified, the string is used in the output of Chain_Test-only patterns. If not specified, the default string is “SCANCHK”. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_CUSTOMER

Specifies the string the tool uses in the CUSTOMER statement in the header of the TITDL output file.

Usage

```
TITDL_CUSTOMER string;
```

Specifies the string the tool uses in the

```
CUSTOMER=string
```

statement in the header of the TITDL output file. If not specified, the default is TEXAS INSTRUMENTS. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_GROUP_TIMING

Setting this keyword to 1 in conjunction with the “[write_patterns -Titdl](#)” command groups like timing information into a single statement.

Usage

```
TITDL_GROUP_TIMING {1 | 0};
```

Setting this keyword to 1 in conjunction with the “[write_patterns -Titdl](#)” command groups like timing information into a single statement. The default is off.

TITDL_KEEP_SCANOUT

Setting this keyword to 1 causes all SCANOUT statements to be issued, even if all of the scanout values for a specific pattern/chain are X.

Usage

```
TITDL_KEEP_SCANOUT {1 | 0};
```

Setting this keyword to 1 causes all SCANOUT statements to be issued, even if all of the scanout values for a specific pattern/chain are X. The default is 0, which causes the tool to not keep scanout statements whose scanout values are all X when writing TITDL format vectors.

TITDL_LIBRARY

Specifies the string that the tool uses in the LIBRARY_TYPE statement in the header of the TITDL output file.

Usage

```
TITDL_LIBRARY string;
```

Specifies the string that the tool uses in the

```
LIBRARY_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is GS20. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_PARTNUM

Specifies the string the tool uses in the TI_PART_NUMBER statement in the header of the TITDL output file.

Usage

```
TITDL_PARTNUM string;
```

Specifies the string the tool uses in the

```
TI_PART_NUMBER=string
```

statement in the header of the TITDL output file. If not specified, the default is F999999. To add this to the specific title block, see the parameter file keyword STIL_TI_TITLE.

TITDL_PSEUDO_PREFIX

Specifies the string that is prepended to the name of the clock pin to create the name of the pseudo clock pin.

Usage

```
TITDL_PSEUDO_PREFIX string;
```

Specifies the string that is prepended to the name of the clock pin to create the name of the pseudo clock pin. If not specified, the default is “pseudo_”.

For example, this keyword changes the default pseudo prefix:

```
TITDL_PSEUDO_PREFIX ps_ ;
```

And this is the result of what the MUX_PIN statement would look like in the TITDL output. The clock pin is called “clock”, and the added pseudo clock pin is called “ps_clock” because of the use of the above parameter file keyword.

```
...
CONNECT P,
VAR =(in1, in2, clock, ps_clock, out1, out2),
DEFPIN=(IN 4, OUT 2);
PERIOD = 100 NS;
CLOCK VAR=(clock),
HOLD0 = 20 NS, HOLD1 = 10 NS, PATTERN = 010;
CLOCK VAR = (ps_clock),
HOLD0 = 60 NS, HOLD1 = 10 NS, PATTERN = 010;
MUX_PIN clock, VAR=(clock, ps_clock);
DELAY VAR = (in1),
OFFSET = 0 NS;
...
```

TITDL_REVISION

Specifies the string that the tool uses in the REVISION statement in the header of the TITDL output file. I

Usage

```
TITDL_REVISION string;
```

Specifies the string that the tool uses in the

```
REVISION=string
```

statement in the header of the TITDL output file. If not specified, the default is 1.00. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_SET_DESCRIPTION

Adds a user-specified string for use as the PATTERN_SET_DESCRIPTION.

Usage

```
TITDL_SET_DESCRIPTION string;
```

When set to a string, this is used in both TITDL files and in STIL files that also use [STIL_TI_TITLE](#) to add the PATTERN_SET_DESCRIPTION statement where the string is used as the description.

TITDL_SETNAME

Specifies the string that the tool uses in the PATTERN_SET_NAME statement in the header of the TITDL output file.

Usage

```
TITDL_SETNAME string;
```

Specifies the string that the tool uses in the

```
PATTERN_SET_NAME=string
```

statement in the header of the TITDL output file. The string can be a maximum of twenty characters. If not specified, the default is the filename with no extension. To add this to the specific title block, use the [STIL_TI_TITLE](#) keyword.

TITDL_SETTYPE

Specifies the string that the tool uses in the PATTERN_SET_TYPE statement in the header of the TITDL output file.

Usage

```
TITDL_SETTYPE string;
```

Specifies the string that the tool uses in the

```
PATTERN_SET_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is SCAN. To add this to the specific title block, use the [STIL_TI_TITLE](#) keyword.

WGL_ADD_LASTX

Specifies for the tool to add an additional vector to the end of the pattern set (when set to 1).

Usage

```
WGL_ADD_LASTX {1 | 0};
```

Specifies for the tool to add an additional vector to the end of the pattern set (when set to 1). This vector is a non-scan vector with all clocks turned off, all output pins set to “X” (not measuring), and all input pins set to the same value of the previous non-scan vector. To deactivate, set this keyword to 0 (the default).

WGL_ALT_BIDI

Changes the syntax of how bidirectional signals are represented in the vector pin list, and the actual states in the vector.

Usage

```
WGL_ALT_BIDI {1 | 0};
```

Changes the syntax of how bidirectional signals are represented in the vector pin list, and the actual states in the vector. The default representation (keyword set to 0) specifies the bidirectional signal name, a colon, then an I or O, for the direction of that pin. There must be two entries for the bidirectional pin, one specifying the input side and one specifying the output side. The vector lists a separate state for each.

Setting this keyword to 1 changes the representation so that the pin list lists only one entry for the bidirectional pin with no direction, but with two states. Every state in the actual vector is currently separated by spaces. With the addition of this WGL_ALT_BIDI 1 setting, the states for the bidirectional pin will be listed side-by-side with no space separation. The input state will be listed first.

WGL_ALT_BIDI set to 0 (default) — The format of the vector order pin list and vector is as follows:

```
pattern test("A","B","C","BiDi1:I","E","F","G","BiDi1:O")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 - 0 1 0 1];
```

WGL_ALT_BIDI set to 1 — The vector order pinlist and vector are changed to the following:

```
pattern test("A","B","C","BiDi1","E","F","G")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 -1 0 1 0];
```

WGL_ALT_VECT_ANN

Specifies alternate annotation statements for beginnings of patterns and vector boundaries when this keyword is set to 1.

Usage

```
WGL_ALT_VECT_ANN {1 | 0};
```

Specifies alternate annotation statements for beginnings of patterns and vector boundaries when this keyword is set to 1. These alternate annotations can help the WGL file be processed by other third party tools. The default is 0.

Note

 Using this keyword can produce a WGL file that you cannot read back into the Tessent tools.

WGL_EDGE_STROBE

Specifies whether to include an X state after an edge strobe.

Usage

```
WGL_EDGE_STROBE {1 | 0};
```

The default value of this keyword is 0. This keyword is in effect only when the strobe_window is set to 0 using the “set strobe_window time” statement in the procedure file.

The WGL output specifies edge strobes by using the edge qualifier on the strobe times specified in timeplate definitions. Additionally, an output value showing the X state appears after the strobe time.

If the strobe_window is set to 0, setting this keyword to 1 eliminates from the WGL output this additional output value showing the X state after the strobe time.

The following is an example of an edge strobe statement without and with using WGL_EDGE_STROBE.

WGL_EDGE_STROBE set to 0 (default):

```
"outpin":=output [0ns:X, 300ns:Q'edge, 350ns:X] ;
```

WGL_EDGE_STROBE set to 1

```
"outpin":=output [0ns:X, 300ns:Q'edge] ;
```

WGL_FULL_CHAIN

When set to 1, this keyword alters the scandell and scanchain blocks of the WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts (number of apply shifts).

Usage

```
WGL_FULL_CHAIN {1 | 0} ;
```

When set to 1, this keyword alters the scandell and scanchain blocks of the WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts (number of apply shifts). The group statements only reflect what is loaded minus the post shift. The scanstate and group have to match. To deactivate, and for backwards compatibility, issue a 0 (false).

Examples follow.

Original WGL File (WGL_FULL_CHAIN 0)

```
...
scandell
  { Group: grp1 }
  { Chain: REG1 }
  "u22"; "u23"; "u10"; "u11"; "u12";
  groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
end
scanchain
  { Group: grp1 }
  REG1_chain[ "sdi", "u12", "u11", !, "u10", "u23", "u22", "sdo" ];
end
...
scanstate
  OREG1_sts000000_000002 := groupgrp1REG1 (XXXXXX);
  IREG1_sts000000_000002 := groupgrp1REG1 (11010);
...
end
...
```

WGL_FULL_CHAIN 1

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
"sdo"] ;
    end
...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (11010);
...
end
...
```

WGL_FULL_SCANGROUP

Creates the Scan sections with the full chain (when set to 1).

Usage

```
WGL_FULL_SCANGROUP {1 | 0};
```

Creates the Scan sections with the full chain (when set to 1). This keyword alters the scanchain, scancell, and scanstate blocks in WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts. Setting the keyword to 0, deactivates this feature.

The following is an example of WGL code when WGL_FULL_SCANGROUP and WGL_FULL_CHAIN are both set to 1:

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12", "u13"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
                    "sdo"];
    end
...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (X10100);
...
end
...
```

WGL_GROUP_PIN

Specifies for the WGL output to create signal groups _PI_, _PO_, and _BIDI_ (if there are bidi's) when set to 1.

Usage

```
WGL_GROUP_PIN {1 | 0};
```

Specifies for the WGL output to create signal groups _PI_, _PO_, and _BIDI_ (if there are bidi's) when set to 1. The WGL code then uses these groups in the pattern statement and in actual patterns for grouping states. To deactivate, set this keyword to 0.

WGL_INV_SC

Removes inversion data from scan chain definitions and specifies scan data as it would appear to be shifted in, instead of parallel deposited.

Usage

```
WGL_INV_SC {1 | 0};
```

Removes inversion data from scan chain definitions and specifies scan data as it would appear to be shifted in, instead of parallel deposited. According to the WGL data model, a WGL file

that uses this option no longer correctly matches the design netlist. However, some third-party tools work faster with the inversion data removed. When set to 0 (default), this keyword leaves the inversion data intact.

WGL_NO_SCANX

Replaces all X scan load values in the WGL pattern output with 0 scan load values.

Usage

```
WGL_NO_SCANX {1 | 0};
```

Replaces all X scan load values in the WGL pattern output with 0 scan load values. When set to 0 (default), this keyword deactivates this feature.

WGL_NOMEASURE_CLOCK

Drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured.

Usage

```
WGL_NOMEASURE_CLOCK {1 | 0};
```

Drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. Setting this keyword to 0 (default), deactivates this feature.

Note that this keyword can cause some WGL readers to have problems reading the file. The tool issues a warning if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow:

```
Parameter file : default
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];

Parameter file : WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];

Parameter file : WGL_TRIM_PULSE 1;
                  WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_ONE_ILLINOIS

Causes the WGL output to have only one scan state defined per pattern for multiple scan chains that share a single scan input pin.

Usage

```
WGL_ONE_ILLINOIS {1 | 0};
```

Causes the WGL output to have only one scan state defined per pattern for multiple scan chains that share a single scan input pin. In other words, this keyword forces the tool to produce a single scan state for chains that share a common scan input pin (set to 1).

The default behavior when this keyword is set to 0 is to have scan states defined for each scan chain, whether or not they share scan input pins. The default for scan states when multiple scan chains share a common scan input is to list the scan states for each cell in each chain, the same as if the chains did not share a scan input.

WGL_PATTERN_NAME

Changes the default name of the WGL pattern block from “Chain_scan_test” to the value of the string.

Usage

```
WGL_PATTERN_NAME string;
```

Changes the default name of the WGL pattern block from “Chain_scan_test” to the value of the string.

WGL_TRIM_LEADING_P

Removes the leading 0ns:P timing event when a channel uses timing variables .

Usage

```
WGL_TRIM_LEADING_P {1 | 0};
```

Removes the leading 0ns:P timing event when a channel uses timing variables (keyword set to 1). For example, the default behavior produces a timeplate channel that looks like this:

```
"pi_sel_smcard" := input[0ns:P, fd_force:S];
```

When WGL_TRIM_LEADING_P is set to 1 in the parameter file, the timeplate channel looks like this:

```
"pi_sel_smcard" := input [fd_force:S];
```

WGL_TRIM_PULSE

Causes the input timing to drop the leading 0 ns force to the off state of the clock.

Usage

```
WGL_TRIM_PULSE {1 | 0};
```

Causes the input timing to drop the leading 0 ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. Setting this keyword to 0 deactivates this feature.

Note that this keyword generates illegal syntax in the WGL file, which may cause some WGL readers to have problems reading the file. The tool issues a warning message if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow.

Parameter file : default

```
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];  
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```

Parameter file : WGL_TRIM_PULSE 1;

```
IOclk := input [ 200ns:S, 300ns:D ];  
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```

Parameter file : WGL_TRIM_PULSE 1;
 WGL_NOMEASURE_CLOCK 1;

```
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_VECTOR_ANN

Adds an annotation statement to be generated in the pattern file to denote the vector type.

Usage

```
WGL_VECTOR_ANN {1 | 0};
```

Adds an annotation statement to be generated in the pattern file to denote the vector type. Setting this keyword to 0 deactivates this feature.

WGL_VERG_ESC

Specifies for buses to be expanded in the declaration sections of the WGL output.

Usage

```
WGL_VERG_ESC {1 | 0};
```

Specifies for buses to be expanded in the declaration sections of the WGL output. Setting this keyword to 0 deactivates this feature.

WGL_VTRAN_PADSC

Pads all scan data to be same length.

Usage

```
WGL_VTRAN_PADSC {1 | 0};
```

Pads all scan data to be same length (keyword set to 1). Padding all scan data to be the same length violates the WGL syntax according to the TDS WGL documentation. However, some third-party tools work better with the scan data padded. Setting this keyword to 0 deactivates this feature.

WGL_WRAP_ANNOTATION

Breaks multi-line annotations into multiple annotation statements.

Usage

```
WGL_WRAP_ANNOTATION {1 | 0};
```

When this keyword is set to 1 or true in the parameter file, then any multi-line annotations are broken into multiple annotation statements, such as:

```
Ann /* this is a long */  
Ann /* multi-line annotation */
```

WGL_WRAP_ANNOTATION

If not present in the parameter file, the default for this keyword is 0 or off.

By default, the tool automatically inserts newlines into annotations when a single line of the annotation exceeds 512 characters, regardless of the setting of WGL_WRAP_ANNOTATION.

Appendix B

HDL Limitations in the Tessent Shell Flow

This appendix describes the HDL limitations which affect functionality in the Tessent Shell Flow. The two tables below summarize the limitations. Full explanations of those limitations along with examples are given in the sections which follow.

The limitations can be categorized by the effect they have on the flow—see the “Effect on Flow” column in [Table B-1](#) on page 4007 and [Table B-2](#) on page 4010.

- **Impacts Compilation and/or Elaboration** — An error will typically be issued by the `read_verilog`, the `read_vhdl`, or the `set_current_design` command.
- **Impacts Design Analysis** — When a limitation in this category is encountered, an error will typically be issued when the tool is tracing through the elaborated circuit. Such errors will typically occur during rules checking and/or in `TS_Bscan` during pad extraction. In most cases tracing in `TS_Ijtag` is not affected. Exceptions are noted in [Table B-1](#) on page 4007 and [Table B-2](#) on page 4010.
- **Impacts Design editing and/or Introspection** — An error will typically be issued when the tool is modifying the design to insert new design components or during execution of an introspection command. Such errors will typically occur in `TS_DesignEditor` or in `TS_Bscan` or `TS_Membist` during design editing.
- **Impacts Design Verification** — An error will typically be issued during test bench generation.
- **Impacts Hierarchical Insertion** — Limitations in this category affect the ability to make needed modifications to designs with child blocks or cores, each with their own workspace. An error will typically be issued when the tool flow performs design rule checking on the interface between child and parent. Such errors will typically occur during design rule checking in the early stages of the flow. Some limitations will result in errors in the child flow and others in the parent flow.

Note

 Language constructs which are not in the RTL-synthesizeable subset are ignored. DC Shell requires that such constructs be surrounded by pragmas; otherwise, an error is generated. If tool edits create a need to update such a construct, for example a Verilog hierarchical name, then a manual edit by the user is required.

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog	4007
HDL Limitations in the Tessent Shell Flow for VHDL	4010

SystemVerilog Interfaces	4013
Multi-dimensional Instantiation of SystemVerilog Interfaces	4013
SystemVerilog Interfaces Cannot Be Edited	4013
Outside Connections on Interface Type Ports Cannot be Intercepted	4013
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type	4014
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block	4014
System Verilog Interfaces in Task or Function Invocation	4015
Nested Modules	4015
SystemVerilog Enumerations	4017
SystemVerilog Enum Methods	4017
SystemVerilog Enum Type in Port Declarations	4017
Complex Port Types for Verilog and SystemVerilog	4018
Access to Ports and Nets of Complex Types	4018
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design	4019
Complex Types in Top-Level Ports at the Chip Level	4019
Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog	4020
SystemVerilog Extern Modules	4020
Unrolled Generate Loops for Verilog and SystemVerilog	4021
Unrolling of Generate Loops Side Effects	4021
Insertion of Custom Logic Which Connects to a Port or Net in an Unrolled Generate Loop	4024
Parameterized Designs for Verilog and SystemVerilog	4025
Parameter Overrides in the Instantiation of the Root Module of a Child Block	4025
Support for the Defparam Construct	4025
Implicit Port Syntax	4026
SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block	4026
SystemVerilog 2009	4027
New Module Insertion in VHDL	4028
Insertion of VHDL Entities with Unconstrained Ports	4028
Insertion of VHDL Entities with Uninitialized Generics	4028
Complex Port Types for VHDL	4029
Access to Ports and Nets of Complex Types	4029
Array of Array and Record Types in Port Declarations in the Root Module of a Design	4029
Complex Types in Top-Level Ports at the Chip Level	4029
Ports with Unconstrained Array Types in the Root Design Entity of a Child Block	4030
Procedures, Functions, and Process Statements for VHDL	4030
Configurations for VHDL	4030
VHDL 2008	4030
Parameterized Designs for VHDL	4032

Generic Overrides in the Instantiation of the Root Design Entity of a Child Block 4032

HDL Limitations in the Tesson Shell Flow for Verilog and SystemVerilog

The following HDL limitations apply to Verilog and SystemVerilog.

The following table lists and categorizes these limitations.

Table B-1. Verilog and SystemVerilog Limitations

Limitation	Effect on Flow	Workaround and Link to Description
SystemVerilog Interfaces		
Multi-dimensional instantiation of SV interfaces is not supported.	Affects tracing and design editing.	Use a single dimension. See “ Multi-dimensional Instantiation of SystemVerilog Interfaces ” on page 4013
SV interfaces cannot be edited.	Affects design editing.	Replace the interface with the equivalent expanded logic. See “ SystemVerilog Interfaces Cannot Be Edited ” on page 4013.
Outside connections on ports declared using an SV interface type cannot be intercepted.	Affects design editing.	Replace interface usage with equivalent expanded logic. See “ Outside Connections on Interface Type Ports Cannot be Intercepted ” on page 4013.
Insertion of an instance of a module containing a port whose type is an SV interface is not supported.	Affects design editing.	Replace interface usage with equivalent expanded logic. See “ Insertion of an Instance of a Module With a SystemVerilog Interface Port Type ” on page 4014.
A port declared in the root module of a child block or core cannot be declared using an SV generic interface type.	Affects design elaboration, hierarchical insertion, and test bench generation.	Use a known interface type. See “ SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block ” on page 4014.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Multiple instantiations of an SV interface cannot be passed as arguments in a single task or function invocation.	Affects RTL quick synthesis needed for checking design rules and ICL extraction.	Uniquify the interface instantiations. See “ System Verilog Interfaces in Task or Function Invocation ” on page 4015.
Nested Modules		
Nested modules are not supported in code which needs to be edited or introspected.	Affects design introspection and editing.	Change the RTL. See “ Nested Modules ” on page 4015.
SystemVerilog Enumerations		
Logic containing SV enum methods such as next, prev, first, and last cannot be traced.	Affects tracing.	Specify the enumerated list values explicitly. See “ SystemVerilog Enum Methods ” on page 4017.
Ports declared using an SV enum type declared in the port declaration itself cannot be intercepted on the inside of the module.	Affects design editing.	Replace the enum type with an integer. See “ SystemVerilog Enum Type in Port Declarations ” on page 4017.
Complex Port Types		
Ports/nets declared using a complex type cannot be addressed in TCL editing and introspection commands.	Affects design introspection and editing.	Replace the complex type usage with equivalent expanded logic. See “ Access to Ports and Nets of Complex Types ” on page 4018.
Unpacked dimensions and other complex types cannot be used in the definitions of root module ports which drive or receive data from inserted DFT logic.	Affects test bench generation.	Use scalars and simple vectors. See “ Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design ” on page 4019.
When boundary scan needs to be inserted into a design, top-level port declarations at the chip level are limited to scalars and simple vectors.	Affects tracing and design editing.	Replace the complex type usage with equivalent expanded logic. See “ Complex Types in Top-Level Ports at the Chip Level ” on page 4019.
Functions, Tasks, and Procedural Statements		

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Tracing through logic containing non-constant function calls, task invocations, and/or procedural statements is not supported.	Affects tracing.	Change the RTL. See “ Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog ” on page 4020.
SystemVerilog Extern Modules		
Extern modules in SV are ignored.	Affects design editing.	Change the RTL. See “ SystemVerilog Extern Modules ” on page 4020.
Unrolled Generate Loops		
Design modifications which require unrolling of generate loops may have undesired side effects.	Affects design editing.	See “ Unrolling of Generate Loops Side Effects ” on page 4021.
Insertion of custom logic which connects to a port or net inside an unrolled generate loop when the insertion of the custom logic and the unrolling are done in different insertion phases (unrolling phase first).	Affects design editing.	See “ Insertion of Custom Logic Which Connects to a Port or Net in an Unrolled Generate Loop ” on page 4024.
Parameterized Designs		
For a parameter declared in the root module of a child block or core which is used in logic that will be edited during insertion, such a parameter cannot be overridden in the instantiation of the child block.	Affects hierarchical insertion.	Use a fixed value in place of the parameter or set the default to the same value as the override. See “ Parameter Overrides in the Instantiation of the Root Module of a Child Block ” on page 4025.
Defparams whose paths are not relative to the current scope are not supported.	Affects design compilation / elaboration.	Move the defparam so that the specified path is relative to the current scope or use the parameter override construct. See “ Support for the Defparam Construct ” on page 4025.
Implicit Port Syntax		

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Implicit port syntax (i.e., <code>.*</code>) cannot be used in the instantiation of a child block or core.	Affects hierarchical insertion.	Use expanded syntax. See “ SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block ” on page 4026.
SystemVerilog 2009		
Certain limitations exist for a small subset of constructs specific to SystemVerilog 2009.	Affects design editing.	Avoid the unsupported constructs if they need to be updated to reflect edits during DFT insertion. See “ SystemVerilog 2009 ” on page 4027.

HDL Limitations in the Tessent Shell Flow for VHDL

The following HDL limitations apply to VHDL.

The following table lists and categorizes these limitations.

Table B-2. VHDL Limitations

Limitation	Effect on Flow	Workaround and Link to Description
New Module Insertion		
Insertion of an instance of a VHDL entity containing a port declared using an unconstrained array type is not supported.	Affects design editing.	Always use constrained array types for ports which are arrays in VHDL entities to be inserted into the design. See “ Insertion of VHDL Entities with Unconstrained Ports ” on page 4028.
Insertion of an instance of a VHDL entity containing uninitialized generics is not supported.	Affects design editing.	Always initialize any generics in VHDL entities to be inserted into the design. See “ Insertion of VHDL Entities with Uninitialized Generics ” on page 4028.
Complex Port Types		

Table B-2. VHDL Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Ports/nets declared using a complex type cannot be addressed in TCL editing and introspection commands.	Affects design introspection and editing.	Replace the complex type usage with equivalent expanded logic. See “ Access to Ports and Nets of Complex Types ” on page 4029.
The array of array and record constructs cannot be used in the definitions of root module ports which drive or receive data from inserted DFT logic.	Affects test bench generation.	Use simple vectors. See “ Array of Array and Record Types in Port Declarations in the Root Module of a Design ” on page 4029.
When boundary scan needs to be inserted into a design, top-level port declarations at the chip level are limited to scalars and simple vectors.	Affects tracing and design editing.	Replace the complex type usage with equivalent expanded logic. See “ Complex Types in Top-Level Ports at the Chip Level ” on page 4029.
A port declared in the root module of a child block or core cannot be declared using an unconstrained array type.	Affects design elaboration, hierarchical insertion and test bench generation.	Use a constrained array type. See “ Ports with Unconstrained Array Types in the Root Design Entity of a Child Block ” on page 4030.
Procedures, Functions, and Process Statements		
Tracing through logic containing procedure calls, non-constant function calls, and/or process statements is not supported.	Affects tracing.	See “ Procedures, Functions, and Process Statements for VHDL ” on page 4030.
Configurations		
Format preservation is not done when editing VHDL configurations. Such edits are typically needed when uniquifying an entity/architecture pair which appears in a configuration.	Affects design editing.	Avoid the use of pragmas in configurations as they will not be preserved when the configuration needs to be edited. See “ Configurations for VHDL ” on page 4030.
When there are multiple configurations for the same entity, only the elaborated / active one can be edited. The inactive configurations may need manual updates to avoid parsing errors later in the flow.	Affects design editing.	Manually update inactive configurations when needed. See “ Configurations for VHDL ” on page 4030.

Table B-2. VHDL Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
VHDL 2008		
VHDL 2008 is not fully supported.	Affects design compilation / elaboration and design editing.	Avoid the unsupported VHDL 2008 constructs. See “ VHDL 2008 ” on page 4030.
Parameterized Designs		
For a generic declared in the root module of a child block or core which is used in logic that will be edited during insertion, such a generic cannot be overridden in the instantiation of the child block.	Affects hierarchical insertion.	Use a fixed value in place of the generic or set the default to the same value as the override. See “ Generic Overrides in the Instantiation of the Root Design Entity of a Child Block ” on page 4032.

SystemVerilog Interfaces

Note the following when using SystemVerilog Interfaces.

Multi-dimensional Instantiation of SystemVerilog Interfaces.....	4013
SystemVerilog Interfaces Cannot Be Edited.....	4013
Outside Connections on Interface Type Ports Cannot be Intercepted.....	4013
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type	4014
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block 4014	
System Verilog Interfaces in Task or Function Invocation	4015

Multi-dimensional Instantiation of SystemVerilog Interfaces

Tesson tools do not support multi-dimensional instantiations of a SystemVerilog interface. Connections in module port lists which utilize such interface instantiations cannot be traced or modified.

The following example illustrates a multi-dimensional instantiation of a SystemVerilog interface:

```
interface intfA;  
...  
endinterface  
  
module top ( input in1, output o1 );  
...  
intfA my_intf_array [3:0][4:0](); // interface instantiation  
...  
endmodule
```

SystemVerilog Interfaces Cannot Be Edited

Tesson tools cannot modify the definition of a SystemVerilog interface.

Outside Connections on Interface Type Ports Cannot be Intercepted

Tesson tools cannot intercept outside connections on ports which are declared using a SystemVerilog interface type.

This limitation will affect boundary scan insertion for cases in which pad cell keyports are declared using a SystemVerilog interface type. It will also affect the insertion of custom logic.

The following example illustrates the use of a SystemVerilog interface type in the declaration of a module port:

```
interface intfA;  
...  
endinterface  
  
module joe ( intfA p1 );  
...  
endmodule
```

An outside connection on *p1* cannot be intercepted.

Insertion of an Instance of a Module With a SystemVerilog Interface Port Type

Tessent tools cannot insert an instance of a module which has a port declared with a SystemVerilog interface type.

SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block

In a bottom-up flow SystemVerilog generic interface port declarations cannot be used in the root module of a child block or core.

The following example illustrates a generic interface port:

```
module mem1 (interface pins, input clk);  
...  
endmodule
```

System Verilog Interfaces in Task or Function Invocation

Multiple arguments which are defined as instantiations of the same SV interface type cannot be passed in a single task or function invocation. For example:

```
interface my_intf #( parameter SZ1 = 1 );
    wire w1[SZ1-1: 0];
    wire w2[SZ1-1: 0];

    // some logic around w1,w2 etc...
endinterface

my_intf #(10) intf_obj_1;
my_intf #(20) intf_obj_2;

...
// Single function,task call using objects of same interface
    my_func (intf_obj_1, intf_obj_2, <other_arguments>);
...
...
```

The work around is to uniquify the instantiations of the interface as follows:

```
interface my_intf #( parameter SZ1 = 1 );
    wire w1[SZ1-1: 0];
    wire w2[SZ1-1: 0];

    // some logic around w1,w2 etc...
endinterface

interface my_intf_1 #( parameter SZ1 = 1 );
    wire w1[SZ1-1: 0];
    wire w2[SZ1-1: 0];

    // some logic around w1,w2 etc...
endinterface

my_intf #(10) intf_obj_1;
my_intf_1 #(20) intf_obj_2;

...
// Single function,task call using objects of different interfaces
    my_func (intf_obj_1, intf_obj_2, <other_arguments>);
...
...
```

Nested Modules

Nested modules are not supported in SystemVerilog code which needs to be edited or introspected.

An example of a nested module is as follows:

```
module my_core (input wire clk);
    submod i1 (...);
    module submod (...);
    ...
endmodule
endmodule
```

SystemVerilog Enumerations

Note the following when using SystemVerilog enumerations.

SystemVerilog Enum Methods 4017

SystemVerilog Enum Type in Port Declarations 4017

SystemVerilog Enum Methods

Tesson tools cannot trace a signal through logic written using SystemVerilog enum methods such as next, prev, first, and last.

In the following example the return values of the first and last methods are used as alternatives in a case generate statement:

```
module joe ( input in1, output out1 );
  typedef enum {S[2]} states_t;
  parameter states_t p1 = S1;
  case(p1)
    p1.first : begin
      assign out1 = p1.first;
    end
    p1.last : begin
      assign out1 = p1.last;
    end
  endcase
endmodule
```

The signal *out1* cannot be traced through the above case generate statement.

SystemVerilog Enum Type in Port Declarations

Tesson tools cannot intercept ports which are declared using a SystemVerilog enum type. This limitation may affect the insertion of custom logic.

The following example illustrates a port declaration with an enum type:

```
module top( input enum {stop, go} in1, input clk, output out1 );
  reg r1;
  always_ff @(posedge clk)
    if( in1 == stop )
      r1 <= 1;
    else
      r1 <= 0;
  assign out1 = r1;
endmodule // top
```

The port *in1* cannot be intercepted, that is, the internal net *in1* cannot be disconnected from the port *in1* and connected to a new driver.

Complex Port Types for Verilog and SystemVerilog

Note the following when using complex port types.

Access to Ports and Nets of Complex Types	4018
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design.....	4019
Complex Types in Top-Level Ports at the Chip Level	4019

Access to Ports and Nets of Complex Types

Ports and nets of the following types cannot be introspected and cannot appear in editing commands:

- User-defined types
- Structs (packed and unpacked)
- Unions
- Packed arrays of more than one dimension
- Unpacked arrays of one or more dimensions

The above limitation puts constraints on the types of connections which can be made with custom logic.

The following example illustrates a user-defined type:

```
module top();
    typedef logic [7:0] my_vec_type;
    my_vec_type w1, w2; // variables declared to be of type my_vec_type
endmodule // top
```

The following example illustrates a structure:

```
module top();
    struct {
        logic a;
        logic b;
    } my_struct;
    assign my_struct.a = 1'b0;
endmodule // top
```

Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design

Unpacked dimensions and other complex types (see previous section for a list) cannot be used in the definitions of root module ports (including those in child blocks or cores and the chip-level design) which drive or receive data from inserted DFT logic.

An example of an unpacked dimension is the following:

```
input cb_flow_type [18:0];
```

Unpacked dimensions always appear to the right of the port identifier.

Complex Types in Top-Level Ports at the Chip Level

The insertion of boundary scan by Tesson tools imposes certain restrictions on top-level port declarations. In Verilog and SystemVerilog designs top-level port declarations at the chip level are limited to one packed dimension and zero unpacked dimensions, that is scalars and simple vectors.

The following example illustrates a module port declaration with two packed dimensions (which is not allowed):

```
module top( input [1:0] [7:0] in1, output out1 );
endmodule
```

The following example illustrates a module port declaration with one packed dimension and one unpacked dimension (which is not allowed):

```
module top( input [1:0] in1 [7:0], output out1 );
endmodule
```

Packed dimensions are those which appear to the left of the port identifier; unpacked dimensions appear to the right of the port identifier.

Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog

Tessent tools cannot trace a signal through logic written using a non-constant function call, a task invocation or a procedural statement.

SystemVerilog Extern Modules [4020](#)

SystemVerilog Extern Modules

Tessent tools cannot modify the ports lists of any extern module declarations in SystemVerilog designs.

The following example illustrates an extern module declaration:

```
// Prototype of car
extern module car ( input wire in1, output wire out1 );

// Definition of car: the .* syntax tells the compiler to get
// the port list from the prototype
module car (*.*);
endmodule
```

Unrolled Generate Loops for Verilog and SystemVerilog

Note the following when using generate loops.

Unrolling of Generate Loops Side Effects	4021
Insertion of Custom Logic Which Connects to a Port or Net in an Unrolled Generate Loop	4024

Unrolling of Generate Loops Side Effects

Tesson tools need to unroll generate loops for certain types of editing.

This type of transformation in Verilog design files can have undesired side effects. Unrolled generate loops in Verilog design files are, by default, written out as a series of if generate blocks. There is one if generate block for each value of the genvar index in the original generate loop. Text inside the loop such as compiler directives, inactive generate regions, and comments is not preserved. An option is available to preserve such text. In that case the unrolled loops are written out as a series of loop generate blocks of size one. There is one loop generate block for each value of the genvar index in the original generate loop. This is in contrast to how unrolled generate loops in edited SystemVerilog design files are written out: if generate blocks are always used to perform the unrolling and text for compiler directives, inactive generate regions, and comments is preserved. The reason for the different handling is due to lack of support in simple Verilog for localparam declarations inside if generate blocks - those are allowed in SystemVerilog. The localparam construct is needed for format preservation (see below). When the unrolling is done using if generate blocks, hierarchical paths (for example, paths in SDC constraints) which reference objects inside generate loops in Verilog design files will still work in most tools. When the unrolling is done using loop generate blocks of size 1, those hierarchical paths will be invalid without manual intervention. Timing scripts generated by Tesson TimingGen tool will account for the use of loop generate blocks of size one; however, user-generated timing scripts will not be updated and may be invalid after unrolling.

As mentioned above, you have control over the unrolling strategy which is used. The unrolling strategy is set via the `-preserve_paths_for_unrolled_v2001_loops` option in the `set_insertion_options` command. If the value is set to OFF, user-generated timing scripts may be affected.

The example below shows how unrolling is performed in Verilog 2001 design files for both values of the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command:

```
wire [1:0] in1, in2;
generate
genvar i;
for( i = 0; i < 2; i++ )
begin : joe
`ifdef TECHNOLOGY_32NM
    DUT i0 ( .A(in1[i]), .B(in2[i]), ... );
`else
    DUT1 i0 ( .A(in1[i]), .B(in2[i]), ... );
`endif
end for
end generate
```

Here is what is written after unrolling when the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command is set to OFF:

```
wire [1:0] in1, in2;
genvar i;
for( i = 0; i <= 0 ; i++ ) begin : \joe[0]
`ifdef TECHNOLOGY_32NM
    DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
for( i = 1; i <= 1; i++ ) begin : \joe[1]
`ifdef TECHNOLOGY_32NM
    DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
```

Generate loops of size 1 are used to perform the unrolling. The compiler directives and the inactive region are preserved. Instance *i0* inside block *\joe[0]* must be referenced (for example in SDC constraints) using the following hierarchical path:

```
\joe[0] [0]/i0
```

Here is what is written after unrolling when the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command is set to ON (assuming that the ` ifdef region is inactive):

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
    DUT1 i0 ( .A(in1[0]), .b(in2[0]), ... );
end
if(1) begin : \joe[1]
    DUT1 i0 ( .A(in1[1]), .b(in2[1]), ... );
end
```

If generate blocks are used to perform the unrolling. The compiler directives and the inactive region are removed. Instance *i0* inside block *\joe[0]* can be referenced (for example, in SDC constraints) using either of the following hierarchical paths:

```
\joe[0] /io
joe[0]/io
```

If we treat the same example as a SystemVerilog design file, then here is what is written after unrolling, regardless of the setting of the *-preserve_paths_for_unrolled_v2001_loops* option in the *set_insertion_options* command:

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
localparam i = 0;
`ifdef TECHNOLOGY_32NM
    DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end
if(1) begin : \joe[1]
localparam i = 1;
`ifdef TECHNOLOGY_32NM
    DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
    DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end
```

If generate blocks are used to perform the unrolling. Since localparam declarations are allowed inside if generate blocks in SystemVerilog, the inactive region can be preserved. Without the localparam, the reference to the variable “i” would generate an error.

In summary the pros and cons of setting the *-preserve_paths_for_unrolled_v2001_loops* to ‘off’ (SDC paths are not preserved; however, text format, comments, and compiler directives are preserved) are as follows:

- **Pros** — Compiler directives, inactive regions, and comments are preserved inside unrolled generate loops in Verilog design files.
- **Cons** — User generated timing scripts which reference objects inside unrolled loops may not work. Manual intervention may be required to fix hierarchical paths in such scripts.

Insertion of Custom Logic Which Connects to a Port or Net in an Unrolled Generate Loop

If custom logic is used to make connections to a net or port inside an unrolled generate loop, the escaped names used for the unrolled loop labels in the edited design file must be used in the custom logic.

For example, if the following path is used before the design is first edited:

A/B [0] /P1

then after loop unrolling, the required syntax is as follows:

A/\B [0] /P1

Note the “\” character and the space character after the “]” character.

Parameterized Designs for Verilog and SystemVerilog

Note the following when using parameterized designs.

Parameter Overrides in the Instantiation of the Root Module of a Child Block.....	4025
Support for the Defparam Construct.....	4025

Parameter Overrides in the Instantiation of the Root Module of a Child Block

In a bottom-up flow parameter overrides cannot be used in the instantiation of the root module of a child block or core when the parameters being overridden are used in logic that will be edited during insertion. For example, such a parameter could not be used as the loop count of a generate loop in which memories are instantiated.

Support for the Defparam Construct

Support for the defparam construct is limited to those for which the specified path is relative to the current scope.

For example:

```
module joe( input jPort );
    bar b1();
    bar b2();
    defparam b1.f1.p = 6;
    ...
endmodule

module bar( input bPort );
    foo f1();
    foo f2();
    ...
endmodule

module foo( input fPort );
    parameter p = 3;
    ...
endmodule
```

Implicit Port Syntax

Note the following when using implicit port syntax.

SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block 4026

SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block

In a bottom-up flow SystemVerilog implicit port syntax (i.e., `.*`) cannot be used in the instantiation of the root module of a child block or core.

The following example illustrates the use of implicit port syntax:

```
module top( input in1, output out1 );
...
joe i1( .* );
endmodule

module joe( input in1, output out1 );
...
endmodule
```

SystemVerilog 2009

Tesson Shell supports reading and writing of all of the SV2009 constructs. However, a few constructs cannot be modified in Tesson Shell. Those are as follows:

- Hierarchical references starting from parent hierarchy
- Coverage constructs
 - covergroup/endgroup
- Class constructs
- Program block constructs
 - program/endprogram
- Assertion constructs
 - assert
 - assert property
 - property/endproperty
 - clocking/endclocking
 - sequence/endsequence
- Checker constructs
 - checker/endchecker
- Bind constructs

For example, if a net is renamed in a [move_connections](#) command and it appears in any of those constructs, that update will not be reflected in the construct when the file containing the construct is written out.

New Module Insertion in VHDL

Note the following when inserting VHDL entities.

Insertion of VHDL Entities with Unconstrained Ports [4028](#)

Insertion of VHDL Entities with Uninitialized Generics [4028](#)

Insertion of VHDL Entities with Unconstrained Ports

Insertion of VHDL entities containing ports defined using unconstrained array types is not supported.

See the section entitled “[Ports with Unconstrained Array Types in the Root Design Entity of a Child Block](#)” on page 4030 for an example of an unconstrained array type.

Insertion of VHDL Entities with Uninitialized Generics

Insertion of VHDL entities containing uninitialized generics is not supported. The following is an example:

```
entity Joe is
    generic ( N : INTEGER );
    port( in1 : in std_logic );
end entity;
```

Complex Port Types for VHDL

Note the following when using complex port types.

- Access to Ports and Nets of Complex Types 4029
Array of Array and Record Types in Port Declarations in the Root Module of a Design
4029
Complex Types in Top-Level Ports at the Chip Level 4029
Ports with Unconstrained Array Types in the Root Design Entity of a Child Block .. 4030

Access to Ports and Nets of Complex Types

Net and port objects declared using an array of array or a record type cannot be introspected and cannot appear in editing commands.

The above limitation puts constraints on the types of connections which can be made with custom logic.

Array of Array and Record Types in Port Declarations in the Root Module of a Design

Array of array and record types cannot be used in the definitions of root module ports (including those in child blocks or cores and the chip-level design) which drive or receive data from inserted DFT logic.

The following example illustrates the use of an array of array in the declaration of a port:

```
package pck is
    type myArray is array( 1 downto 0 ) of bit_vector( 0 to 0 );
end;

entity john is
    port( in1 : in myArray; out1 : out bit );
end john;
```

Complex Types in Top-Level Ports at the Chip Level

The insertion of boundary scan by Tesson tools imposes certain restrictions on top-level port declarations. In VHDL designs top-level port declarations at the chip level must be scalars or simple vectors.

Ports with Unconstrained Array Types in the Root Design Entity of a Child Block

Ports declared in the root module of a child block or core cannot be declared using an unconstrained array type. The following is an example of an unconstrained array type:

```
type JOE is array (INTEGER range <>) of std_logic;
```

Procedures, Functions, and Process Statements for VHDL

Tessent tools cannot trace a signal through logic written using a non-constant function call, a procedure invocation or a process statement.

Configurations for VHDL

MemoryBIST and boundary scan insertion are not fully supported in configured designs.

Format is not preserved when configurations need to be edited. Therefore, pragmas and other comments will not be preserved in edited design files which contain configurations.

In addition when there are multiple configurations for the same entity, only the elaborated / active one can be edited. The inactive configurations may need manual updates to avoid parsing errors later in the flow. For example, if a component referenced in a configuration is unqualified, then any inactive configuration(s) for the same entity may need manual updates.

VHDL 2008

Tessent Shell supports the following VHDL 2008 constructs:

- New VHDL 2008 generate block syntax
- Unconstrained element support in arrays and records
- Logical reduction operator
- Expressions which appear directly in a port map (e.g., a boolean or logical expression whose evaluated result is connected to the named port)
- Reading of output ports
- Simplified sensitivity lists
- C-style block comments
- Operations involving both an array and a scalar (e.g., “A_bus and A_sel”; A_sel is 1 bit and it applies to all bits of A_bus)

- Extensions to bit string literals
- Slices in array aggregates

VHDL 2008 constructs which are not in the above list are currently not supported.

Parameterized Designs for VHDL

Note the following when using parameterized designs.

Generic Overrides in the Instantiation of the Root Design Entity of a Child Block . . . 4032

Generic Overrides in the Instantiation of the Root Design Entity of a Child Block

In a bottom-up flow generic overrides cannot be used in the instantiation of the root design entity of a child block or core if those generics are used in logic that will be edited during insertion. Furthermore default values must be specified for such generics.

The following example illustrates a VHDL entity with two generics - one with a default value and one without:

```
entity joe is
    generic (g1:integer; g2:integer :=0);
    ...
end;
```

The generic *g1* is defined without a default value. The generic *g2* is defined with a default value of 0. The above entity cannot be used as the root of a child block if the generic *g1* is used in logic that will be edited during insertion. If *g1* had been specified with a default value, then entity *joe* could be used as the root of a child block only if it is instantiated in the parent block without overrides for the generics.

Appendix C

Getting Help

There are several ways to get help when setting up and using Tesson software tools. Depending on your need, help is available from documentation, online command help, and Mentor Graphics Support.

The Tesson Documentation System	4033
Mentor Support Services.....	4034

The Tesson Documentation System

At the center of the documentation system is the InfoHub that supports both PDF and HTML content. From the InfoHub, you can access all locally installed product documentation, system administration documentation, videos, and tutorials. For users who want to use PDF, you have a PDF bookcase file that provides access to all the installed PDF files.

For information on defining default HTML browsers, setting up browser options, and setting the default PDF viewer, refer to the “[Documentation Options](#)” in the *Mentor Documentation System* manual.

You can access the documentation in the following ways:

- **Shell Command** — On Linux platforms, enter **mgcdocs** at the shell prompt or invoke a Tesson tool with the -manual invocation switch.
- **File System** — Access the Tesson InfoHub or PDF bookcase directly from your file system, without invoking a Tesson tool. For example:

HTML:

```
firefox $MGC_DFT/docs/infohubs/index.html
```

PDF

```
acroread $MGC_DFT/docs/pdfdocs/_bk_tesson.pdf
```

- **Application Online Help** — You can get contextual online help within most Tesson tools by using the “help -manual” tool command. For example:

> help dofile -manual

This command opens the appropriate reference manual at the “dofile” command description.

Mentor Support Services

Mentor provides a range of industry-leading support services that keep design teams productive and up-to-date with Mentor products.

A Mentor support contract includes the following:

- **Software Updates** — Get the latest releases and product enhancements to keep your environment current.
- **Mentor Graphics Support Center** — Access our online knowledge base, personalized to your Mentor products.
- **Support Forums** — Learn, share, and connect with other Mentor users.
- **Technical Support** — Collaborate with Mentor support engineers to solve complex design challenges.
- **Regular Communications** — Receive the latest knowledge base articles and announcements for your Mentor products.
- **Mentor Ideas** — Share ideas and vote for your favorites to shape future products.

More information is available here:

<https://support.mentor.com>

If your site is under a current support contract, but you do not have a Support Center login, register today:

<https://support.mentor.com/register>

Index

— A —

A rules, 2666
add_ambiguous_paths, 94
add_atpg_constraints, 96
add_atpg_functions, 100
add_bist_capture_range, 103
add_black_box, 105
add_browser_data, 109, 226
add_capture_handling, 111
add_cell_constraints, 115
add_clocks, 128
add_display_data, 226
add_display_instances, 228
add_edt_blocks, 241
add_false_paths, 246
add_fault_sites, 253
add_faults, 255
add_iddq_exceptions, 279
add_input_constraints, 286
add_lfsr_connections, 295
add_lfsr_taps, 297
add_lfsrs, 299
add_lists, 302
add_nottest_points, 319
add_output_masks, 319
add_primary_inputs, 321
add_primary_outputs, 324
add_processors, 326
add_read_controls, 329
add_scan_chains, 336
add_scan_groups, 339
add_synchronous_clock_group, 356
add_tied_signals, 358
add_write_controls, 362
ALL_FIXED_CYCLES, 3948
ALL_FLATTEN_TIMING, 3948
ALL_MIN_SCAN_LOAD, 3951
ALL_NO_CYCLE_OPT, 3951
ALL_NO_LOOP, 3952

ALL_NO_PATTERN_TYPE, 3952
ALL_TIME_RESOLUTION, 3954
analyze_bus, 367
analyze_control_signals, 393
analyze_drc_violation, 396
analyze_fault, 398
analyze_graybox, 404
analyze_restrictions, 408
analyze_simulation_mismatches, 412
analyze_wrapper_cells, 425

— B —

BIST rules, 2686
Brules, 2686

— C —

C rules, 2687
Clock cone, 2700, 2707
Clocks, 1910
close_visualizer, 455
Commands
 add_ambiguous_paths, 94
 add_atpg_constraints, 96
 add_atpg_functions, 100
 add_bist_capture_range, 103
 add_black_box, 105
 add_browser_data, 109, 226
 add_capture_handling, 111
 add_cell_constraints, 115
 add_cell_models, 122
 add_chain_masks, 124
 add_clocks, 128
 add_display_data, 226
 add_display_instances, 228
 add_edt_blocks, 241
 add_false_paths, 246
 add_fault_sites, 253
 add_faults, 255
 add_iddq_exceptions, 279
 add_input_constraints, 286

add_lfsr_connections, 295
add_lfsr_taps, 297
add_lfsrs, 299
add_lists, 302
add_nottest_points, 319
add_output_masks, 319
add_primary_inputs, 321
add_primary_outputs, 324
add_processors, 326
add_read_controls, 329
add_scan_chains, 336
add_scan_groups, 339
add_scan_instances, 341
add_synchronous_clock_group, 356
add_test_points, 358
add_tied_signals, 358
add_write_controls, 362
analyze_bus, 367
analyze_control_signals, 393
analyze_drcViolation, 396
analyze_fault, 398
analyze_graybox, 404
analyze_restrictions, 408
analyze_simulation_mismatches, 412
analyze_wrapper_cells, 425
close_visualizer, 455
compress_patterns, 459
create_capture_procedures, 469
create_flat_model, 496
create_initialization_patterns, 502
create_patterns, 521
delete_atpg_constraints, 542
delete_atpg_functions, 544
delete_bist_capture_range, 546
delete_black_box, 547
delete_browser_data, 549
delete_capture_handling, 551
delete_capture_procedures, 553
delete_cell_constraints, 556
delete_cell_library, 558
delete_cell_models, 559
delete_clocks, 562
delete_design, 575
delete_display_data, 590
delete_display_instances, 592
delete_edt_blocks, 594
delete_edt_configurations, 597
delete_false_paths, 599
delete_fault_sites, 602
delete_faults, 606
delete_iddq_exceptions, 622
delete_input_constraints, 625
delete_lfsr_connections, 634
delete_lfsr_taps, 635
delete_lfsrs, 636
delete_lists, 637
delete_multicycle_paths, 641
delete_nofaults, 646
delete_nonscan_instances, 649
delete_nottest_points, 651
delete_output_masks, 654
delete_patterns, 656
delete_primary_inputs, 663
delete_primary_outputs, 665
delete_processors, 667
delete_read_controls, 669
delete_scan_chains, 674
delete_scan_groups, 676
delete_scan_instances, 677
delete_synchronous_clock_group, 685
delete_test_points, 687
delete_tied_signals, 689
delete_write_controls, 691
diagnose_failures, 692
display_diagnosis_report, 696
dofile, 702
echo, 718
find_design_names, 737
get_attribute_value_list, 755
get_clock_option, 766, 771
get_multiprocessing_option, 946
get_pattern_cycle_count, 954
help, 1041
identify_redundant_faults, 1060
mark_display_instances, 1136
open_visualizer, 1169
order_patterns, 1171
read_cell_library, 1202
read_cpf, 1208
read_failures, 1220, 1818

read_fault_sites, 1228
read_faults, 1238
read_modelfile, 1257
read_patterns, 1261
read_procfile, 1271
read_sdc, 1276
read_sdf, 1283
read_upf, 1286, 1288
read_visualizer_preferences, 1299
report_aborted_faults, 1379
report_atpg_constraints, 1382
report_atpg_functions, 1383
report_atpg_timing, 1384
report_bist_capture_range, 1396
report_black_box, 1397
report_bus_data, 1401
report_bypass_chains, 1404
report_capture_handling, 1407
report_capture_procedures, 1409
report_cell_constraints, 1415
report_clock_domains, 1424
report_clocks, 1431
report_compactor_connections, 1436
report_control_signals, 1453
report_display_instances, 1482
report_drc_rules, 1485
report_edt_blocks, 1509
report_edt_instances, 1521
report_edt_pins, 1529
report_environment, 1532
report_external_simulator, 1534
report_failures, 1535
report_false_paths, 1539
report_fault_sites, 1547
report_faults, 1555
report_feedback_paths, 1569
report_flattener_rules, 1572
report_gates, 1574
report_graybox_statistics, 1599
report_id_stamp, 1608
report_iddq_exceptions, 1610
report_input_constraints, 1618
report_lfsr_connections, 1636
report_lfsrs, 1637
report_lists, 1640
report_loops, 1642
report_measure_cycles, 1647
report_mismatch_sources, 1665
report_multicycle_paths, 1678
report_multiprocessing_options, 1684
report_nofaults, 1686
report_nonscan_cells, 1688
report_nonscan_models, 1690
report_notest_points, 1691
report_output_masks, 1693
report_power_data, 1709
report_power_metrics, 2319
report_primary_inputs, 1717
report_primary_outputs, 1719
report_procedures, 1721
report_processors, 1725
report_read_controls, 1728
report_resources, 1733
report_scan_cells, 1737
report_scan_chains, 1745
report_scan_groups, 1755
report_scan_volume, 1764
report_seq_transparent_procedures, 1768
report_statistics, 1785
report_synchronous_clock_groups, 1804
report_test_logic, 1810
report_test_points, 1811
report_test_stimulus, 1818
report_testability_data, 1827
report_tied_signals, 1828
report_timeplate, 1830
report_wrapper_cells, 1837
report_write_controls, 1841
reset_au_faults, 1848
reset_bypass_chains, 1850
reset_compactor_connections, 1851
reset_design, 1852
reset_di_faults, 1853
reset_state, 1857, 1861
select_display_instances, 1902
set_abort_limit, 1904
set_atpg_fill, 1906
set_atpg_limits, 1907
set_atpg_timing, 1910
set_attribute_value, 1918

set_au_analysis, 1918
set_bist_chain_test, 1927
set_bist_trace, 1932
set_bus_handling, 1937
set_bus_simulation, 1940
set_bypass_chains, 1942
set_capture_clock, 1945
set_capture_handling, 1948
set_capture_procedures, 1950
set_cell_library_options, 1953
set_chain_test, 1959
set_checkpointing_options, 1963
set_clock_off_simulation, 1977
set_clock_option, 1979
set_clock_restriction, 1982
set_compactor_connections, 1989
set_contention_check, 1996
set_current_design, 2025
set_current_edt_block, 2014
set_current_edt_configuration, 2017
set_decision_order, 2025
set_design_include_directories, 2033
set_design_macros, 2038
set_diagnosis_options, 2039, 2045, 2049
set_dofile_abort, 2063
set_drc_handling, 2065
set_driver_restriction, 2073
set_edt_instances, 2077
set_edt_mapping, 2081
set_edt_options, 2084
set_edt_pins, 2095
set_external_capture_options, 2110
set_external_simulator, 2113
setfails_report, 2115
set_fault_mode, 2117
set_fault_sampling, 2119
set_fault_subclass_analysis, 2121
set_fault_type, 2124
set_flattener_rule_handling, 2133
set_gate_level, 2136
set_gate_report, 2138
set_gzip_options, 2172
set_iddq_checks, 2178
set_internal_fault, 2202
set_internal_name, 2203
set_io_mask, 2204
set_latch_handling, 2205
set_learn_report, 2222
set_lfsrs, 2225
set_list_file, 2229
set_logfile_handling, 2230
set_loop_handling, 2234
set_makrotest, 2251
set_multiprocessing_options, 2267
set_net_dominance, 2275
set_net_resolution, 2277
set_number_shifts, 2278
set_observation_point, 2280
set_output_masks, 2282
set_pattern_buffer, 2288
set_pattern_classification, 2290
set_pattern_source, 2301
set_pattern_type, 2310
set_possible_credit, 2318
set_procedure_cycle_checking, 2329
set_procfile_name, 2333
set_ram_initialization, 2338
set_random_atpg, 2339
set_random_clocks, 2340
set_random_patterns, 2342
set_shadow_check, 2366
set_skewed_load, 2379
set_split_capture_cycle, 2380
set_stability_check, 2382
set_static_learning, 2390
set_system_mode, 2392
set_test_logic, 2398
set_tied_signals, 2423
set_timing_exceptions_handling, 2425
set_tla_loop_handling, 2431
set_trace_report, 2438
set_transient_detection, 2442
set_transition_holdpi, 2444
set_visualizer_preferences, 2450
set_xclock_handling, 2464
set_z_handling, 2466
set_zhold_behavior, 2468
setup_test_point_insertion, 2410, 2423
simulate_patterns, 2475
stil2mgc, 71

unmark_display_instances, 2497
update_implicit_detections, 2506
write_atpg_setup, 2512
write_design, 2526, 2634
write_edt_files, 2543
write_failures, 2560
write_fault_sites, 2565
write_faults, 2569
write_flat_model, 2581
write_history, 2586
write_loops, 2586
write_modelfile, 2588
write_paths, 2589
write_patterns, 2589
write_primary_inputs, 2610
write_primary_outputs, 2611
write_procedure_testbench, 2612
write_procfile, 2616
write_scan_order, 2618
write_visualizer_dofile, 2634
write_visualizer_preferences, 2636
write_window_contents, 2637

Compactor
specifying, 2090

Compactor
Xpress compactor masking, 2090
Xpress compactor masking, 2091

compress_patterns, 459

create_capture_procedures, 469

create_flat_model, 496

create_initialization_patterns, 502

create_patterns, 521

CTL_ONE_PAT, 3954

CTL_STIL_0, 3955

CTL_STIL_1, 3955

CTL_TRIM_PULSE, 3988

— D —

D rules, 2732, 2747

delete_atpg_constraints, 542

delete_atpg_functions, 544

delete_bist_capture_range, 546

delete_black_box, 547

delete_browser_data, 549

delete_capture_handling, 551

delete_capture_procedures, 553

delete_cell_constraints, 556

delete_cell_library, 558

delete_clocks, 562

delete_design, 575

delete_display_data, 590

delete_display_instances, 592

delete_edt_blocks, 594

delete_edt_configurations, 597

delete_false_paths, 599

delete_fault_sites, 602

delete_faults, 606

delete_iddq_exceptions, 622

delete_input_constraints, 625

delete_lfsr_connections, 634

delete_lfsr_taps, 635

delete_lfsrs, 636

delete_lists, 637

delete_multicycle_paths, 641

delete_nofaults, 646

delete_notest_points, 651

delete_output_masks, 654

delete_patterns, 656

delete_primary_inputs, 663

delete_primary_outputs, 665

delete_processors, 667

delete_read_controls, 669

delete_scan_chains, 674

delete_scan_groups, 676

delete_synchronous_clock_group, 685

delete_tied_signals, 689

delete_write_controls, 691

Design Rule Checking, how to display violations, 1485

Design rules checking
basic troubleshooting, 2641
BIST rules, 2686
clock rules, 2687
data rules, 2732, 2747
EDT rules, 2895
extra rules, 2775
general rules, 2829
procedure rules, 2916
RAM rules, 2666
reporting gate data, 2648
scannability rules, 2984

setting gate data, 2648
setting gate level, 2646
setting rule handling, 2641
timing rules, 3032
trace rules, 2994
with ATPG analysis, 2641
diagnose_failures, 692
display_diagnosis_report, 696
dofile, 702
DRC messages
oscillation limitation, 3053
other DRC messages, 3052
RAM summary results and test capability, 3053
transparent capture handling analysis, 3052
DRC, how to display violations, 1485

— E —

E rules, 2775
Effect cone, 2700
EMPTY, 2652, 2675, 2722, 2746, 2792, 2834, 2913, 2946, 2989, 3016, 3040

— F —

Faults
timing-critical, 1910
FTDL_TEST_NAME, 3957
FTDL_TEST_NUM, 3957
FTDL_TEST_SUFFIX, 3957
find_design_names, 737
FTDL parameter file example, 3958
FTDL_DESIGNER, 3956
FTDL_MAX_PAT_BLOCK, 3956
FTDL_REVISION, 3956
FTDL_ZMODE_MES, 3957

— G —

G rules, 2829
get_attribute_value_list, 755
get_clock_option, 766, 771
get_multiprocessing_option, 946
get_pattern_cycle_count, 954
group timing information, 3992
Group TITDL timing information, 3992

— H —

help, 1041

Hold_pi, 1910

— I —

identify_redundant_faults, 1060
Instance name, 3965
Introduction, 53

— K —

K rules, 2895

Keywords
ALL_FIXED_CYCLES, 3948
ALL_FLATTEN_TIMING, 3948
ALL_MIN_SCAN_LOAD, 3951
ALL_NO_CYCLE_OPT, 3951
ALL_NO_LOOP, 3952
ALL_NO_PATTERN_TYPE, 3952
ALL_TIME_RESOLUTION, 3954
CTL_ONE_PAT, 3954
CTL_STIL_0, 3955
CTL_STIL_1, 3955
CTL_TRIM_PULSE, 3988
FTDL_DESIGNER, 3956
FTDL_MAX_PAT_BLOCK, 3956
FTDL_REVISION, 3956
FTDL_TEST_NAME, 3957
FTDL_TEST_NUM, 3957
FTDL_TEST_SUFFIX, 3957
FTDL_ZMODE_MES, 3957
SIM_COMPARE_X, 3960
SIM_EARLY_RELEASE, 3963
SIM_EARLY_RELEASE_TIME, 3963
SIM_INCLUDE, 3965
SIM_INSTANCE_NAME, 3965
SIM_KEEP_PATH, 3965
SIM_MASK_FILE, 3966
SIM_MISCOMPARE_LIMIT, 3967
SIM_MODULE_INCLUDE, 3968
SIM_NO_MEASURE_IN, 3969
SIM_SHIFT_DEBUG, 3972
SIM_STATUS_MSG, 3974
SIM_TIMEPLATE_COM, 3975
SIM_TOP_NAME, 3975
SIM_VECTOR_COMM, 3975
SIM_VECTYPE_SIGNAL, 3976
STIL_2005_SCAN, 3978
STIL_MIN_QUOTE, 3981

STIL_NOMESURE_CLOCK, 3982
STIL_PAT_ANN, 3982
STIL_PAT_CMT, 3983
STIL_PAT_LAB, 3983
STIL_SCAN_ANN1, 3984
STIL_STRUCTURAL, 3985
STIL_TI_TITLE, 3988
STIL_VECTOR_ANN, 3989
TITDL_CHAIN_SETTYPE, 3991
TITDL_CUSTOMER, 3991
TITDL_GROUP_TIMING, 3992
TITDL_KEEP_SCANOUT, 3992
TITDL_LIBRARY, 3992
TITDL_PARTNUM, 3992
TITDL_PSEUDO_PREFIX, 3993
TITDL_REVISION, 3993
TITDL_SETNAME, 3994
TITDL_SETTYPE, 3994
WGL_ADD_LASTX, 3995
WGL_ALT_BIDI, 3995
WGL_ALT_VECT_ANN, 3996
WGL_EDGE_STROBE, 3996
WGL_FULL_CHAIN, 3997
WGL_FULL_SCANGROUP, 3999
WGL_GROUP_PIN, 3999
WGL_INV_SC, 4000
WGL_NOMEASURE_CLOCK, 4000
WGL_ONE_ILLINOIS, 4001
WGL_PATTERN_NAME, 4001
WGL_TRIM.LEADING_P, 4002
WGL_TRIM_PULSE, 4002
WGL_VECTOR_ANN, 4003
WGL_VERG_ESC, 4003
WGL_VTRAN_PADSC, 4003

— M —

mark_display_instances, 1136
Mask_po, 1910
Masking scan chains, 124

— N —

newlink Interface99999, 3456
Non-scan initialization values, 339

— O —

open_visualizer, 1169

order_patterns, 1171
Oscillation limitation, 3053

— P —

P rules, 2916

Parameter file keywords

FTDL_DESIGNER, 3956
FTDL_MAX_PAT_BLOCK, 3956
FTDL_REVISION, 3956
FTDL_TEST_NAME, 3957
FTDL_TEST_NUM, 3957
FTDL_TEST_SUFFIX, 3957
FTDL_ZMODE_MES, 3957

— R —

RAM

rules checking, 2666

RAM summary results and test capability, 3053

read_cell_library, 1202
read_cpf, 1208
read_failures, 1220, 1818
read_fault_sites, 1228
read_faults, 1238
read_modelfile, 1257
read_patterns, 1261
read_procfile, 1271
read_sdc, 1276
read_sdf, 1283
read_upf, 1286
read_verilog, 1288
read_visualizer_preferences, 1299
report_aborted_faults, 1379
report_atpg_constraints, 1382
report_atpg_functions, 1383
report_atpg_timing, 1384
report_bist_capture_range, 1396
report_black_box, 1397
report_bus_data, 1401
report_bypass_chains, 1404
report_capture_handling, 1407
report_capture_procedures, 1409
report_cell_constraints, 1415
report_clock_domains, 1424
report_clocks, 1431
report_compactor_connections, 1436

report_drc_rules, 1485
report_edt_blocks, 1509
report_edt_instances, 1521
report_edt_pins, 1529
report_environment, 1532
report_external_simulator, 1534
report_failures, 1535
report_false_paths, 1539
report_fault_sites, 1547
report_faults, 1555
report_feedback_paths, 1569
report_flattener_rules, 1572
report_gates, 1574
report_graybox_statistics, 1599
report_id_stamp, 1608
report_iddq_exceptions, 1610
report_input_constraints, 1618
report_lfsr_connections, 1636
report_lfsrs, 1637
report_lists, 1640
report_loops, 1642
report_measure_cycles, 1647
report_mismatch_sources, 1665
report_multicycle_paths, 1678
report_multiprocessing_options, 1684
report_nofaults, 1686
report_nonscan_cells, 1688
report_notest_points, 1691
report_output_masks, 1693
report_power_data, 1709
report_power_metrics, 2319
report_primary_inputs, 1717
report_primary_outputs, 1719
report_procedures, 1721
report_processors, 1725
report_read_controls, 1728
report_resources, 1733
report_scan_cells, 1737
report_scan_chains, 1745
report_scan_groups, 1755
report_scan_volume, 1764
report_seq_transparent_procedures, 1768
report_synchronous_clock_groups, 1804
report_test_stimulus, 1818
report_testability_data, 1827
report_tied_signals, 1828
report_timeplate, 1830
report_write_controls, 1841
reset_au_faults, 1848
reset_bypass_chains, 1850
reset_compactor_connections, 1851
reset_design, 1852
reset_di_faults, 1853
reset_state, 1857, 1861

— S —

Scan chain trace rules, 2994
Scan chains
 masking, 124
Scannability rules checking, 2984
SDF file, 1283
select_display_instances, 1902
set_abort_limit, 1904
set_atpg_fill, 1906
set_atpg_limits, 1907
set_atpg_timing, 1910
set_attribute_value, 1918
set_au_analysis, 1918
set_bist_chain_test, 1927
set_bist_trace, 1932
set_bus_handling, 1937
set_bus_simulation, 1940
set_bypass_chains, 1942
set_capture_clock, 1945
set_capture_handling, 1948
set_capture_procedures, 1950
set_cell_library_options, 1953
set_chain_test, 1959
set_checkpointing_options, 1963
set_clock_off_simulation, 1977
set_clock_option, 1979
set_clock_restriction, 1982
set_compactor_connections, 1989
set_contention_check, 1996
set_current_design, 2025
set_current_edt_block, 2014
set_current_edt_configuration, 2017
set_decision_order, 2025
set_design_include_directories, 2033
set_design_macros, 2038
set_diagnosis_options, 2039, 2045, 2049

set_dofile_abort, 2063
set_drc_handling, 2065
set_driver_restriction, 2073
set_edt_options, 2084
set_edt_pins, 2095
set_external_capture_options, 2110
set_external_simulator, 2113
setfails_report, 2115
set_fault_mode, 2117
set_fault_sampling, 2119
set_fault_subclass_analysis, 2121
set_fault_type, 2124
set_flattener_rule_handling, 2133
set_gate_level, 2136
set_gate_report, 2138
set_gzip_options, 2172
set_iddq_checks, 2178
set_internal_fault, 2202
set_internal_name, 2203
set_io_mask, 2204
set_learn_report, 2222
set_lfsrs, 2225
set_list_file, 2229
set_logfile_handling, 2230
set_loop_handling, 2234
set_macrotest, 2251
set_multiprocessing_options, 2267
set_net_dominance, 2275
set_net_resolution, 2277
set_number_shifts, 2278
set_observation_point, 2280
set_output_masks, 2282
set_pattern_buffer, 2288
set_pattern_classification, 2290
set_pattern_source, 2301
set_pattern_type, 2310
set_possible_credit, 2318
set_procedure_cycle_checking, 2329
set_procfile_name, 2333
set_ram_initialization, 2338
set_random_atpg, 2339
set_random_clocks, 2340
set_random_patterns, 2342
set_shadow_check, 2366
set_skewed_load, 2379
set_split_capture_cycle, 2380
set_stability_check, 2382
set_static_learning, 2390
set_system_mode, 2392
set_tied_signals, 2423
set_timing_exceptions_handling, 2425
set_tla_loop_handling, 2431
set_trace_report, 2438
set_transient_detection, 2442
set_transition_holdpi, 2444
set_visualizer_preferences, 2450
set_xclock_handling, 2464
set_z_handling, 2466
set_zhold_behavior, 2468
setup_test_point_insertion, 2410
SIM_COMPARE_X, 3960
SIM_EARLY_RELEASE, 3963
SIM_EARLY_RELEASE_TIME, 3963
SIM_INCLUDE, 3965
SIM_INSTANCE_NAME, 3965
SIM_KEEP_PATH, 3965
SIM_MASK_FILE, 3966
SIM_MISCOMPARE_LIMIT, 3967
SIM_MODULE_INCLUDE, 3968
SIM_NO_MEASURE_IN, 3969
SIM_SHIFT_DEBUG, 3972
SIM_STATUS_MSG, 3974
SIM_TIMEPLATE_COM, 3975
SIM_TOP_NAME, 3975
SIM_VECTOR_COMM, 3975
SIM_VECTYPE_SIGNAL, 3976
simulate_patterns, 2475
Simulation mismatches
 for ChainTest patterns, 3966
Slack margin, 1910
STIL_2005_SCAN, 3978
STIL_MIN_QUOTE, 3981
STIL_NOMESURE_CLOCK, 3982
STIL_PAT_ANN, 3982
STIL_PAT_CMT, 3983
STIL_PAT_LAB, 3983
STIL_SCAN_ANN1, 3984
STIL_STRUCTURAL, 3985
STIL_VECTOR_ANN, 3989
stil2mgc script, 71

Suffix
 adding to the test block, 3957

— T —

T rules, 2994
test block
 adding a suffix, 3957
Test Procedure File, definition of, 59, 66
Test structures
 identification interactions, 1425
TI title block
 changing with STIL_TI_TITLE, 3988
Timing-aware ATPG
 setting up, 1910
Timing-critical faults, 1910
TITDL, 3992
TITDL_CHAIN_SETTYPE, 3991
TITDL_CUSTOMER, 3991
TITDL_GROUP_TIMING, 3992
TITDL_KEEP_SCANOUT, 3992
TITDL_LIBRARY, 3992
TITDL_PARTNUM, 3992
TITDL_PSEUDO_PREFIX, 3993
TITDL_REVISION, 3993
TITDL_SETNAME, 3994
TITDL_SETTYPE, 3994
Title block
 changing with STIL_TI_TITLE, 3988
Trace rules checking, 2994
Tran sparent_capture cells, 2745
Transparent capture handling analysis, 3052

— U —

unmark_display_instances, 2497
update_implicit_detections, 2506

— V —

Verilog
 instance name, 3965

— W —

WGL_ADD_LASTX, 3995
WGL_ALT_BIDI, 3995
WGL_ALT_VECT_ANN, 3996
WGL_EDGE_STROBE, 3996
WGL_GROUP_PIN, 3999
WGL_INV_SC, 4000

WGL_NOMEASURE_CLOCK, 4000
WGL_ONE_ILLINOIS, 4001
WGL_PATTERN_NAME, 4001
WGL_TRIM.LEADING_P, 4002
WGL_TRIM_PULSE, 4002
WGL_VECTOR_ANN, 4003
WGL_VERG_ESC, 4003
WGL_VTRAN_PADSC, 4003
write_design, 2526, 2634
write_edt_files, 2543
write_failures, 2560
write_fault_sites, 2565
write_faults, 2569
write_flat_model, 2581
write_history, 2586
write_loops, 2586
write_modelfile, 2588
write_paths, 2589
write_patterns, 2589
write_procedure_testbench, 2612
write_procfile, 2616
write_visualizer_dofile, 2634
write_visualizer_preferences, 2636
write_window_contents, 2637

Third-Party Information

For information about third-party software included with this release of Tesson products, refer to the [*Third-Party Software for Tesson Products*](#).

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an "Order"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation, setup files and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Except for Software that is embeddable ("Embedded Software"), which is licensed pursuant to separate embedded software terms or an embedded software supplement, Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 4.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. BETA CODE.

- 3.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively “Beta Code”), which may not be used without Mentor Graphics’ explicit authorization. Upon Mentor Graphics’ authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.
- 3.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer’s use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer’s evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 3.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer’s feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 3.3 shall survive termination of this Agreement.

4. RESTRICTIONS ON USE.

- 4.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except for Embedded Software that has been embedded in executable code form in Customer’s product(s), Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer’s employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Customer acknowledges that Software provided hereunder may contain source code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such source code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, disassemble, reverse-compile, or reverse-engineer any Product, or in any way derive any source code from Software that is not provided to Customer in source code form. Log files, data files, rule files and script files generated by or for the Software (collectively “Files”), including without limitation files containing Standard Verification Rule Format (“SVRF”) and Tcl Verification Format (“TVF”) which are Mentor Graphics’ trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 4.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use, or as permitted for Embedded Software under separate embedded software terms or an embedded software supplement. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer’s employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 4.3. Customer agrees that it will not subject any Product to any open source software (“OSS”) license that conflicts with this Agreement or that does not otherwise apply to such Product.
- 4.4. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise (“Attempted Transfer”), without Mentor Graphics’ prior written consent and payment of Mentor Graphics’ then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics’ prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics’ option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer’s permitted successors in interest and assigns.
- 4.5. The provisions of this Section 4 shall survive the termination of this Agreement.

5. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics’ then current End-User Support Terms located at <http://supportnet.mentor.com/supportterms>.
6. **OPEN SOURCE SOFTWARE.** Products may contain OSS or code distributed under a proprietary third party license agreement, to which additional rights or obligations (“Third Party Terms”) may apply. Please see the applicable Product documentation (including license files, header files, read-me files or source code) for details. In the event of conflict between the terms of this Agreement

(including any addenda) and the Third Party Terms, the Third Party Terms will control solely with respect to the OSS or third party code. The provisions of this Section 6 shall survive the termination of this Agreement.

7. LIMITED WARRANTY.

- 7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."
 - 7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
8. **LIMITATION OF LIABILITY.** TO THE EXTENT PERMITTED UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. THIRD PARTY CLAIMS.

- 9.1. Customer acknowledges that Mentor Graphics has no control over the testing of Customer's products, or the specific applications and use of Products. Mentor Graphics and its licensors shall not be liable for any claim or demand made against Customer by any third party, except to the extent such claim is covered under Section 10.
- 9.2. In the event that a third party makes a claim against Mentor Graphics arising out of the use of Customer's products, Mentor Graphics will give Customer prompt notice of such claim. At Customer's option and expense, Customer may take sole control of the defense and any settlement of such claim. Customer WILL reimburse and hold harmless Mentor Graphics for any LIABILITY, damages, settlement amounts, costs and expenses, including reasonable attorney's fees, incurred by or awarded against Mentor Graphics or its licensors in connection with such claims.
- 9.3. The provisions of this Section 9 shall survive any expiration or termination of this Agreement.

10. INFRINGEMENT.

- 10.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
- 10.2. If a claim is made under Subsection 10.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
- 10.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) OSS, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such OSS; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
- 10.4. THIS SECTION 10 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE,

SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

11. TERMINATION AND EFFECT OF TERMINATION.

- 11.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.
- 11.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination of this Agreement and/or any license granted under this Agreement, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
12. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and European Union ("E.U.") and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local, E.U. and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any E.U., U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.
13. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.
14. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
15. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 15 shall survive the termination of this Agreement.
16. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America or Japan, and the laws of Japan if Customer is located in Japan. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply, or the Tokyo District Court when the laws of Japan apply. Notwithstanding the foregoing, all disputes in Asia (excluding Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
17. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
18. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Any translation of this Agreement is provided to comply with local legal requirements only. In the event of a dispute between the English and any non-English versions, the English version of this Agreement shall govern to the extent not prohibited by local law in the applicable jurisdiction. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.